TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIKKO NIEMINEN

JAVASCRIPT-POHJAISTEN OHJELMISTOKEHYSTEN SUORITUSKYKY ALUSTARIIPPUMATTOMASSA MOBIILIKEHITYKSESSÄ

Diplomityö

Tarkastaja: prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 6. huhti-kuuta 2016

TIIVISTELMÄ

MIKKO NIEMINEN: JAVASCRIPT-POHJAISTEN OHJELMISTOKEHYSTEN SUORITUSKYKY ALUSTARIIPPUMATTOMASSA MOBIILIKEHITYKSESSÄ
Tampereen teknillinen yliopisto
Diplomityö, 45 sivua, 11 liitesivua
Kesäkuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Pervasive Systems
Tarkastaja: professori Tommi Mikkonen

Avainsanat: hybridisovellus, natiivisovellus, hybridinatiivi, web-sovellus, suorituskyky, Facebook, React, React Native, Google, AngularJS, Drifty, Ionic, Apache Cordova, HTML5, JavaScript, alustariippumaton ohjelmistokehitys, mobiilikehitys, WebView

Usean mobiilikäyttöjärjestelmän suosio markkinoilla luo haasteita mobiilisovellusten kehitykseen. Sovelluksesta joudutaan toteuttamaan useampia alustakohtaisia versioita hyödyntäen toisistaan eroavia teknologioita. Tehdyn työn uudelleenhyödynnettävyys on vähäistä, sillä alustojen välillä ei ole yhteistä alustan tukemaa natiiviksi sovellukseksi käännettävää ohjelmointikieltä.

Tässä diplomityössä tutkittiin alustariippumattomien JavaScript-ohjelmointikieleen pohjautuvien Ionic- ja React Native -ohjelmistokehysten suorituskykyä ja testattiin niiden soveltuvuutta vaihtoehdoksi natiiville mobiilisovellukselle. Soveltuvuuden arviointia varten toteutettiin hybridi- ja hybridinatiivisovellukset edellä mainituilla ohjelmistokehyksillä Android-käyttöjärjestelmälle. Testisovelluksia arvioitiin mittaamalla sovelluksen vaatimia resursseja sekä käyttäjätestin avulla. Työssä tutkittiin myös miten sovelluskehittäjä voi toteutustavoillaan vaikuttaa alustariippumattoman sovelluksen suorituskykyyn.

Tutkimuksen mukaan alustariippumattomia ohjelmistokehyksiä on mahdollista hyödyntää natiivisovelluksen sijaan sovellukselle asetetuista vaatimuksista riippuen. Ionic-testisovelluksen resurssien käyttö oli selkeästi hybridinatiivia React Native -sovellusta korkeampaa. Myös käyttäjätesteissä React Native -sovellus suoriutui paremmin, mutta myös Ionic-sovelluksen suorituskyvyn nähtiin olevan riittävä useimmissa tilanteissa.

Alustariippumattomien teknologioiden soveltuvuus mobiilikehitykseen natiivisovellusten kilpailijoiksi ei tämän työn perusteella ole enää kyseenalainen. Oleellinen kysymys toteutustapaa valittaessa on, missä tilanteissa natiivisovellus kykenee tuomaan riittävästi lisäarvoa verrattuna alustariippumattoman ratkaisun tarjoamaan kustannustehokkuuteen. Tämän työn tulosten perusteella natiivisovelluksen toteuttaminen on edelleen perusteltua tilanteissa, joissa sovelluksen käyttöliittymä vaatii ehdotonta suorituskykyä tai sovelluksen ominaisuudet vaativat laitteistolta tukea ominaisuuksiin, joita tarjolla olevat alustariippumattomat toteutukset eivät vielä tue. Lähitulevaisuudessa markkinoilla olevien laitteiden suorituskyvyn parantuessa ja JavaScript-virtuaalikoneiden kehittyessä erot natiivien- ja alustariippumattomien toteutustapojen välillä tulevat todennäköisesti pienenemään alustariippumattomien teknologioiden eduksi.

ABSTRACT

MIKKO NIEMINEN: PERFORMANCE OF JAVASCRIPT-BASED FRAMEWORKS IN CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT
Tampere University of Technology
Master of Science Thesis, 45 pages, 11 Appendix pages
June 2016
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Professor Tommi Mikkonen

Keywords: hybrid application, native application, web-application, performance, Facebook, React, React Native, Google, AngularJS, Drifty, Ionic, Apache Cordova, HTML5, JavaScript, cross-platform development, mobile development, WebView

Today there are multiple popular mobile operating systems on the market, which presents different challenges to mobile application development. Multiple different implementations of a single application are needed to provide support to all required platforms. Reuse of existing work is challenging because a common natively supported programming language, which can be compiled to a native application, does not yet exist.

In this thesis the performance of two JavaScript-based platform independent frameworks was measured to find the most appropriate method to implement applications in a multiplatform development use case. Two separate Android test applications were implemented with Ionic and React Native frameworks to serve as a platform for the measurements. The performance of these test applications was determined by measuring the resource utilization of both applications and with user testing. Means of impacting the application performance during implementation using the mentioned frameworks were also studied.

The results of this thesis show that platform independent JavaScript frameworks present a qualified option for application development depending on the requirements. An Ionic application uses distinctively more resources than its React Native counterpart. Likewise in the user testing the React Native application managed to overcome the performance of the Ionic application. However most of the users perceived the performance of the Ionic application to be sufficient in most situations.

According to this thesis the suitable use of cross-platform development tools in place of purely native applications is no longer questionable. A more relevant question is in which cases do native applications provide enough benefits to justify the significantly higher development costs. The development of a native application might still be justified in situations when a high level of graphical performance is needed or the cross-platform alternatives do not offer support for required hardware interfaces. In the near future the performance gap of native and cross-platform applications will also likely diminish as the available devices and JavaScript engines evolve.

ALKUSANAT

Tämä diplomityö suoritettiin Eatech Oy:lle tukemaan mobiilisovellusten toteutustapoihin liittyvien päätösten tekoa osana sovellusten myyntiä ja kehitystä. Haluan kiittää Eatech Oy:ta erityisesti vapaista käsistä työn aiheen valinnassa, tarjotuista koulutuksista sekä herkullisista torstaipullista ja pohjattomasta kahvivarannosta. Haluan kiittää myös kollegoitani, jotka haastoivat aktiivisesti ajatuksiani mobiilisovellusten kehityksestä.

Erityisesti haluan kiittää myös professori Tommi Mikkosta työni ja ajatusteni ohjaamisesta, terävistä ja asiallisista kommenteista sekä kärsivällisestä ja positiivisesta asenteesta läpi koko diplomityöprosessin.

Tampereella, 10.05.2016

Mikko Nieminen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

Android debug bridge (ADB)	Komentorivityökalu Android-laitteiden etähallintaan. [30]
Alustariippumattomuus	Useimmiten mobiilisovellusten kehityksessä käytetty termi, joka kuvaa saman sovelluksen kykyä toimia useammalla käyttöjärjestelmällä ilman käyttöjärjestelmäkohtaista toteutusta.
AngularJS	JavaScript-ohjelmistokehys, joka mahdollistaa web-sovellusten toteuttamisen MVC-mallia mukaillen.
Apache Cordova	Alustariippumaton ohjelmistokehys, joka mahdollistaa hybridisovellusten kehittämisen ja paketoimisen laitteeseen asennettavaksi sovellukseksi.
Crosswalk	Intel Corporationin kehittämä WebView-näkymä, jonka tarkoituksena on parantaa hybridisovellusten suorituskykyä ja testattavuutta.
CSS	Cascading Style Sheets. Useimmiten HTML-merkintäkielen kanssa käytettävä tyylittelykieli, jonka avulla on mahdollista muokata HTML-elementtien ulkoasua.
Document Object Model	Dokumenttioliomalli. Useimmiten HTML-merkintäkielellä kuvatun dokumentin kuvaamiseen käytetty puumainen malli.
GPS	Global Positioning System. Satelliittipaikannusjärjestelmä, joka on laajasti käytössä nykyaikaisissa älypuhelimissa. Mahdollistaa laitteen tarkan paikannuksen maailmanlaajuisesti.
HTML	Hyper-Text Markup Language. Merkintäkieli, jota käytetään useimmiten verkkosivustojen esittämiseen puumaisena rakenteena. Internet-selaimet mallintavat merkintäkielen avulla dokumentin paremmin ymmärrettäväksi graafiseksi käyttöliittymäksi.
Hybridinatiivi	Sovellus, joka yhdistää yleisesti natiivisovellusten ja web-sovellusten toteutuksessa käytettyjä teknologioita. Sovelluksen logiikka on esimerkiksi toteutettu JavaScript-ohjelmointikielellä, mutta käyttöliittymän komponentit ovat natiiveja.
Hybridisovellus	Sovellus, joka hyödyntää web-ympäristöissä yleisesti käytettyjä teknologioita (HTML, CSS, JavaScript) ja paketoi nämä laitteeseen asennettavaksi sovellukseksi natiivisovellusrungon sisään.
Injektointi	Olion antaminen toisen olion riippuvuudeksi riippuvuutta käyttävän olion instanssia luodessa. Injektoitu olio toimii tämän jälkeen palveluna, jonka tarjoamaa toiminnallisuutta kohteena oleva olio käyttää.

Ionic	Ohjelmistokehys hybridisovellusten kehitykseen. [18]
Jatkokehitys	Ohjelmiston alkuperäisen käyttöönoton jälkeen suoritettava ohjelmiston toiminnallisuuden laajentaminen.
JavaScript	Pääasiassa web-ympäristöissä hyödynnettävä komentosarjakieli. [14]
Kehitysympäristö	Kokoelma tietylle ohjelmointikielelle tai alustalle ohjelmistojen kehitykseen suunnattuja työkaluja. Sisältää yleensä esimerkiksi tietyn kielen käännöstyökalut ja emulaattorin kehitettävälle alustalle.
Komentosarjakieli	Ohjelmointikieli, joka mahdollistaa sillä kirjoitetun ohjelman suorittamisen kääntämättä siitä ensin suoritettavaa ohjelmaa. Komentosarjakieltä tulkitaan ajonaikaisesti sille tehdyssä ajoympäristössä.
Laitteistorajapinta	Laitteen tarjoama kokoelma metodeja, joiden avulla laite tarjoaa toimintojaan käytettäväksi esimerkiksi laitteeseen asennetulle sovellukselle. Laite voi esimerkiksi tarjota rajapinnan GPS-anturin tarjoaman datan käyttöön.
Minifiointi	Ohjelmakooditiedostoille suoritettava prosessi, jossa tiedostoista poistetaan ylimääräiset merkit. Toimenpide pienentää tiedoston kokoa ja parantaa suorituskykyä tiedostoa siirtäessä ja lukiessa.
Mobiilisivusto	Verkkosivusto, joka on optimoitu mobiililaitteiden pientä ruutua ja rajoitettua suorituskykyä sekä tiedonsiirtoa varten. Mobiilisivusto ei skaalaudu suurelle ruudulle, vaan tämän rinnalle on usein toteutettu erillinen työpöytäkäyttöön suunnattu verkkosivusto. Vrt. responsiivinen verkkosivusto.
MVC	Lyhenne sanoista model-view-controller. Arkkitehtuurimalli, jossa ohjelmisto jakautuu malliin (model), näkymään (view) ja kontrolleriin (controller). Arkkitehtuurissa malli toteuttaa ohjelmiston tiedon käsittelyn ja ylläpidon, näkymä toteuttaa käyttöliittymän ja kontrolleri huolehtii näkymältä saatujen syötteiden käsittelyn ja välityksen mallille.
Natiivisovellus	Laitealustalle alustan tukemalla ohjelmointikielellä toteutettu sovellus, joka käännetään suoritettavaksi ohjelmaksi.
Niputus	Ohjelmakooditiedostoille suoritettava prosessi, jossa samantyyppiset tiedostot yhdistetään yhdeksi tiedostoksi. Englanninkielinen termi: bundling.
Offline-tila	Tavallisesti verkkoon kytketyn laitteen tila, jossa verkkoyhteys on katkaistu tai sitä ei ole saatavilla.

Ohjelmakoodikanta	Kokoelma lähdekoodia, jota käytetään tietyn suoritettavan ohjelman kääntämiseen.
Push-ilmoitus	Ilmoitus, jonka sovelluksen omistaja voi laukaista ja lähettää käyttäjän laitteeseen, vaikka sovellus, jota viesti koskee, olisi suljettuna
Rajapinta	Kokoelma toimintoja, joiden avulla ohjelmisto tarjoaa muille ohjelmistoille mahdollisuuden hyödyntää tämän ominaisuuksia.
React Native	Facebook:n kehittämä ohjelmistokehys hybridinatiivien sovellusten kehittämiseen JavaScript-ohjelmointikielellä. Pohjautuu React-ohjelmistokehykseen. [22]
Responsiivisuus	Suunnitteluperiaate, jonka mukaan käyttöliittymä skaalautuu laitteen näyttökoolle sopivaksi tarjoten paremman käyttäjäkokemuksen.
Responsiivinen verkkosivusto	Verkkosivusto, jonka käyttöliittymä mukautuu käyttäjän laitteen näyttökoon mukaisesti. Responsiivinen verkkosivusto korvaa tarpeen erilliselle mobiilisivustolle.
Ryömijä	Ohjelma, joka lukee internet-sivustoja ja arvostelee sivustojen sijoituksen ryömijän omistavan organisaation hakukoneessa. Tunnetaan myös nimellä hakurobotti. [29]
Singleton	Luokka, josta on mahdollista luoda kerralla ohjelmaan vain yksi instanssi.
Sisällönjakeluverkko	Hajautettu joukko palvelimia, joiden tarkoituksena on palvella resursseja verkkopalvelun varsinaisen web-palvelimen sijasta. Sisällönjakeluverkon etuna on korkeampi suorituskyky ja saatavuus. Englanninkielinen termi: content delivery network.
Sovelluskauppa	Tietyn käyttöjärjestelmän sovellusten jakeluun tarkoitettu sovellus tai sivusto, jonka kautta on mahdollista ostaa, ladata ja asentaa sovelluksia laitteeseen.
Sovelluskaupan saturaatiopiste	Piste, jonka jälkeen sovelluskauppaan julkaistaville sovelluksille ei enää löydy riittävästi kysyntää markkinoilta.
SPA-sovellus	Web-sovellus, joka koostuu päänäkymästä, jonka sisään ladataan dynaamisesti muuta sisältöä. Mahdollistaa rakenteen, jossa sivua ei jouduta lataamaan kokonaan uudelleen näkymän vaihtuessa. [49]
Syntaksilaajennus	Ohjelmointikielen tavallista syntaksia laajentava kirjoitussyntaksi, joka useimmiten muunnetaan puhtaasti kielen vakiosyntaksiksi ennen varsinaista käännöstä.

Taktiilinen palaute	Tuntoaistiin perustuva palaute, jonka sovellus antaa käyttäjälle merkkinä esimerkiksi syötteen onnistuneesta vastaanotosta.
Toimialue	Internetissä olevien resurssien tai näiden kokoelman yksilöimiseen käytetty sanamuotoinen osoite.
Virtual DOM	React-ohjelmistokehyksen tarjoama perinteisen dokumenttioliomallin abstraktiotaso, johon sovelluksessa toteutetut komponentit mallinnetaan ennen varsinaiseen dokumenttioliomalliin mallintamista.
Web-sovellus	Mobiilisovelluksen kaltaisesti käyttäytyvä useimmiten responsiivinen verkkosivusto, joka sisältää dynaamista toiminnallisuutta.
WebView	Laitealustan internet-selainta emuloiva näkymä sovelluksen sisällä. Mahdollistaa HTML-merkkauskielen tulkitsemisen käyttäjän ymmärtämiksi elementeiksi internet-selaimen tapaan.

1. JOHDANTO

Usean eri käyttöjärjestelmän suosio markkinoilla ja mobiililaitteiden suosion kasvu muihin alustoihin verrattuna on johtanut siihen, että laadukkaiden ja käyttökokemukseltaan miellyttävien mobiilisovellusten kehittämiseen joudutaan käyttämään yhä entistä enemmän resursseja. Käyttäjät odottavat sovelluksen käyttäjäkokemukselta yhä enemmän, sillä perusvaatimukset täyttäviä sovelluksia on tarjolla runsaasti.

Kirjoitushetkellä markkinoita hallitsevilla mobiilikäyttöjärjestelmillä on kullakin omat kehitystyökalunsa ja ohjelmointikielensä, jotka vaativat jokainen omanlaistaan asiantuntemusta. Sovelluskehittäjät, joilta löytyy ammattitaitoa useamman alustan natiivisovelluskehitykseen ja kykenevät työskentelemään ketterästi myös mobiilikehityksen ulkopuolella ovat harvassa. Lisäksi vaikka edellä mainittu kehittäjä löydettäisiin, vaatii useamman alustan kehitys ja ylläpito usein pääomaa moninkertaisesti verrattuna yksittäisen alustan toteutukseen. Perinteisten yksittäiseen alustaan sidottujen natiivisovellusten lisäksi mobiilisovelluksia on mahdollista kehittää alustariippumattomasti perinteisesti web-kehityksessä käytetyillä ohjelmointikielillä ja työkaluilla. Nykyhetkellä vallitsee kuitenkin yleisesti käsitys, että ainoastaan natiivisovelluksella on mahdollista täyttää käyttäjien vaatimukset sovelluksen suorituskyvylle.

Tässä työssä tutkitaan JavaScriptia käyttävien alustariippumattomien ohjelmistokehysten ja natiivisovellusten välisiä eroja suorituskyvyssä sekä kvantitatiivisin että kvalitatiivisin mittauksin. Työn tavoitteena on löytää vastaus siihen mitä toteutusteknologiaa on viisainta käyttää toteutettaessa hyötysovellusta usealle eri mobiilialustalle, kun käytössä olevia resursseja rajataan toteutustiimin koon, ajan ja pääoman suhteen. Työssä esitellään myös vaihtoehtoja tilanteisiin, joissa tiettyä alustariippumatonta teknologiaa ei ole mahdollista tai kustannustehokasta käyttää.

Tämän työn luvussa 2 käsitellään nykypäivän mobiilikehityksen haasteita sekä vaihtoehtoja sovelluksen arkkitehtuuria ja toteutustekniikoita valitessa. Luvussa esitellään kolme nykyhetkellä pääosin käytettyä toteutustapaa: natiivisovellus, web-sovellus ja hybridisovellus. Luvussa 3 käsitellään tässä työssä mitattavien testisovellusten toteutuksiin käytetyt tekniikat: Ionic ja React Native -ohjelmistokehykset. Luvussa 4 esitellään toteutetut testisovellukset sekä mittaustavat, joilla sovellusten väliset erot saatiin selville. Kaikki mittaukset toteutettiin Android-käyttöjärjestelmälle, sillä alustojen välisten erojen sijaan tarkoituksena oli mitata valittujen toteutusteknologioiden välisiä eroja. Luvussa 5 esitellään ja analysoidaan suoritettujen mittausten tulokset ja esitetään näiden pohjalta tehdyt johtopäätökset. Luvussa arvioidaan myös saatujen tulosten luotettavuutta ja käsitellään tulevaisuuden kehitysideoita tälle työlle. Luvussa 6 suoritetaan yhteenveto tehdyistä johtopäätöksistä ja vastataan esitettyyn tutkimuskysymykseen siitä, ovatko alustariippumattomat toteutustekniikat kilpailukykyinen vaihtoehto natiivisovellukselle.

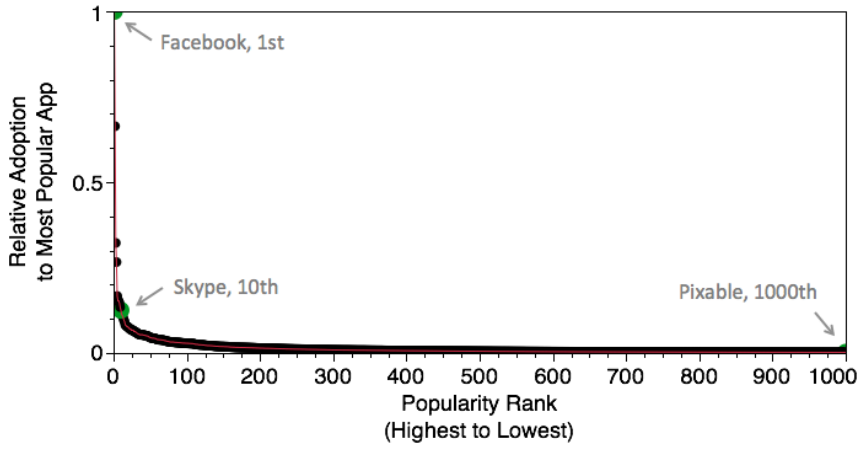
2. MOBIILIKEHITYKSEN TAUSTAA

Tässä luvussa käsitellään, kuinka mobiilimarkkinat ovat muokanneet mobiilikehityksessä käytettäviä teknologisia ratkaisuja ja liiketoimintamalleja. Luvussa esitellään myös korkean tason arkkitehtuuriratkaisut mobiilikehityksessä sekä valitun toteutustavan vaikutus sovelluksen markkinointiin ja jakeluun.

2.1 Mobiilikehitys ja sen haasteet

Mobiilikehityksen ongelmakohtana on jo pitkään ollut usean eri mobiilialustan suosio markkinoilla. Jako on viime vuosina pääosin jakaantunut Googlen Android- ja Apple iOS -alustojen välille näiden kattaessa yhdessä lähes 97 % (2015 Q2) myytyjen mobiililaitteiden markkinoista. Kolmantena merkittävänä alustana voidaan mainita Microsoftin Windows Phone -alusta n. 2,6 % (2015 Q2) osuudellaan. [34]

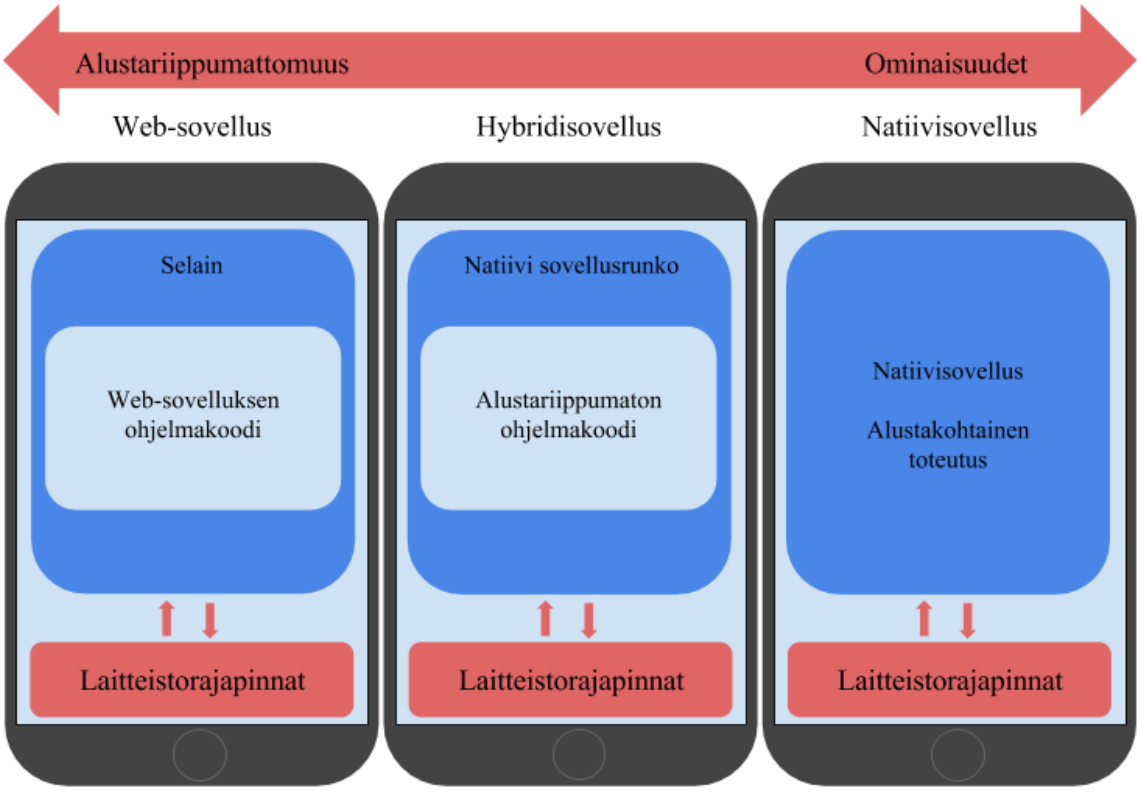
Usean alustan lisäksi ongelmakohtana on mobiilialustojen suosion myötä saavutettu sovellusten saturaatiopiste erityisesti suurimpien alustojen sovelluskaupoissa. Kuvassa 1 käsitellään A. Austinin keräämää tilastoa sovellusten suosion jakautumisesta Applen App Store -sovelluskaupassa [6]. Kuvasta nähdään, että suosituimpaan sovellukseen verrattuna tuhannenneksi suosituin sovellus saavuttaa vain n. 0,2 % osuuden suosituimman sovelluksen asennusmäärästä. Lisäksi sovellusten markkinaosuus on jo jonkin aikaa noudattanut potenssilain käyrää, minkä vuoksi esimerkiksi Apple App Storessa 20 suosituinta sovellusta (n. 0,005 % osuus kaikista sovelluksista) kerää itselleen n. 60 % koko sovelluskaupan sovellusten liikevaihdosta. Käytännössä tämä tarkoittaa sitä, että sovelluskaupoista asennettavien sovellusten määrä on kasvanut niin suureksi, että suosittujen sovellusten joukosta erottuminen on vaikeaa tai lähes mahdotonta. Tämän vuoksi varsinkin tuotteistettavan sovelluksen kehityksestä aiheutuneet kustannukset ovat yhä suurempi riski jo sovelluksen kehitysvaiheessa.



Kuva 1. *Sovellusten suosion jakauma Apple App Storessa [6]*

Uuden sovelluksen mahdollisuudet menestyä sovelluskaupassa riippuivat jonkin aikaa sovelluksen laadusta. Nykyään sovellusten suuren määrän vuoksi suosio riippuu usein ainoastaan tavasta, jolla sovellusta markkinoidaan. Markkinointi voidaan suorittaa esimerkiksi kohdistetulla hakukonemainonnalla, sosiaalisen median näkyvyydellä tai vaihtoehtoisesti ostamalla tuhansia väärennettyjä asennuksia. Suuri määrä väärennettyjä asennuksia lyhyellä ajanjaksolla johtaa siihen, että sovellus nousee hetkellisesti sovelluskaupan suosituimpien sovellusten listalle, jonka kautta se saa näkyvyyttä ja oikeita asennuksia. Samanaikaisesti laadullisesti parempi vastaava sovellus saattaa jäädä kokonaan ilman käyttäjiä. [6]

Useamman eri alustan suosio ja sovellusten suuri määrä pakottaa sovelluskehittäjät harkitsemaan natiivisovelluksen, laitteen selaimessa toimivan responsiivisen web-sovelluksen tai laitteeseen natiivisovelluksen kaltaisesti asennettavan, mutta alustariippumattoman hybridisovelluksen välillä. Valinta on tasapainottelua alustariippumattomuuden ja sovellukselle mahdollisten ominaisuuksien välillä (Kuva 2). Natiivisovellus kykenee hyödyntämään laitteen tarjoamaa laitteistorajapintaa kokonaisuudessaan ja soveltuu täten ominaisuuksiltaan vaativien sovellusten toteutukseen. Web-sovellus ei kykene hyödyntämään laitteistoa suoraan lainkaan, vaan ominaisuudet riippuvat täysin selaimesta, joka alustasta riippuen kykenee tarjoamaan pääsyn esimerkiksi laitteen kameraan tai GPS-sijaintiin. Hybridisovellusta puolestaan ajetaan natiivin sovellusrungon päällä, joka tarjoaa alustasta riippuen useimmiten pääsyn joko kaikkiin tai lähes kaikkiin laitteistorajapintoihin. [2]



Kuva 2. *Sovellustyyppien korkean tason arkkitehtuurit*

2.2 Natiivisovellus

Natiivisovelluksella tarkoitetaan kokonaisuudessaan alustan tukemalla käännettävällä ohjelmointikielellä kirjoitettua ohjelmaa, joka käännetään ja paketoidaan ladattavaksi sovellukseksi kyseisen alustan sovelluskauppaan. Sovellusta ei ole tavallisesti mahdollista käyttää suoraan toisella alustalla, eikä tämän lähdekoodia yleensä mahdollista uudelleenkäyttää toisen alustan sovelluksen kehityksessä. Saman sovelluksen kehitys ja ylläpito usealla alustalla vaatii runsaasti pääomaa, työvoimaa sekä aikaa [15]. Lisäksi mikäli samaa sovellusta toteutetaan useammalle alustalle samanaikaisesti eri teknologioilla eri henkilöiden toimesta, syntyy päällekkäisestä työstä ylimääräisiä kuluja. Vaihtoehtoisesti voidaan toteuttaa vain yksi sovellusversio esimerkiksi iOS-alustalle ja samalla sulkea ulos n. 84 % (2015 Q2) potentiaalisesta asiakaskunnasta [34]. Sovelluksen kehitysvaiheen kannalta toteutus yksittäiselle alustalle on luonnollisesti monialustatoteutukseen verrattuna edullinen vaihtoehto, mutta myyntimahdollisuuksien kannalta merkittävä menetys.

Olettaen, että kutakin alustaa kohden on työstämässä samantasoiset sovelluskehittäjät, voidaan yksittäisen alustan ohjelmakoodin tuottamisen välittömät kustannukset yleensä kertoa toteutettavien alustojen määrällä. Useampaan sovellusversioon sijoitetun pääoman kasvaessa myös paine onnistumiseen kasvaa ja pääoman menetys on merkittävämpi riski. Lisäksi varsinkin pienikokoisissa ohjelmistoyrityksissä useamman alustan toteutusta varten joudutaan usein palkkaamaan ylimääräistä henkilökuntaa, sillä riittävää asiantuntemusta jokaista alustaa kohtaan ei välttämättä löydy yrityksestä valmiina.

Mikäli sovellus halutaan kehittää ainoastaan käyttöliittymältään natiiviksi, on mahdollista hyödyntää esimerkiksi C#-ohjelmointikieleen pohjautuvaa Xamarin-kehitysalustaa tai JavaScriptiin pohjautuvaa React Native -ohjelmistokehystä. Kumpikin mainittu teknologia hyödyntää alustan tarjoamaa rajapintaa natiivien komponenttien piirtoa varten. Tavallisesta natiivisovelluksesta toteutus eroaa kuitenkin siten, että sovelluksen logiikkaosuus on mahdollista kirjoittaa alustasta riippumatta kehyksen tukemalla kielellä. Tämän vuoksi kyseisiä natiiviteknologioita ja esimerkiksi web-teknologioita yhdistäviä toteutustapoja kutsutaan usein myös hybridinatiiveiksi. [54][53][20]

Ohjelmistokehittäjän näkökulmasta puhtaasti natiivien mobiilisovellusten toteuttaminen on ongelmallista. Saman sovelluksen toteuttaminen useammalle alustalle vaatii useamman alustan natiivin ohjelmointikielen sekä työkalujen opettelua. Lisäksi mikäli kutakin alustakohtaista versiota kehittää eri henkilö, eivät kehittäjät kykene juurikaan hyötymään toistensa tekemästä työstä. Kehittäjät eivät myöskään voi kehittää ominaisuuksia helposti ristiin opettelematta kummankin alustan toteutusteknologioita. Projektien resursoinnille saattaa myös aiheutua haasteita erityisesti pienissä ohjelmistoyrityksissä, mikäli kehittäjiä ei ole mahdollista siirtää joustavasti projektista toiseen ilman siirtymästä aiheutuvaa perehtymistä vieraaseen teknologiaan. Mainituista ongelmista huolimatta natiivisovelluksen etuna on kuitenkin laitteistoyhteensopivuus oman alustansa kanssa. Alustojen kehitystyökaluista vastaavat useimmiten suoraan kyseistä käyttöjärjestelmää kehittävät tahot,

mikä takaa sen, että laitteistossa tarjolla olevat uudet ominaisuudet ovat ensimmäisenä tarjolla natiivisovellusten kehitykseen.

2.3 Web-sovellus

Natiivisovelluksen sijaan on mahdollista toteuttaa tavallista verkkosivustoa muistuttava web-sovellus, jota suoritetaan laitteen selaimessa. Useimmiten web-sovellusten toteuttamiseen käytetään SPA-arkkitehtuurin mahdollistavaa ohjelmistokehystä, jolloin web-sovelluksen käyttöliittymä saadaan muistuttamaan läheisemmin mobiilisovellusta ilman ylimääräisiä sivulatauksia [49]. Web-sovellus on lisäksi mahdollista toteuttaa reponsiivisesti tai vaihtoehtoisesti erillisenä mobiilisivustona työpöytäversion rinnalle. Responsiivisuudella tarkoitetaan sitä, että toteutetun web-sovelluksen käyttöliittymä skaalautuu automaattisesti käyttäjän laitteen näytön koon mukaan tarjotakseen paremman käyttäjäkokemuksen. Responsiivisen suunnittelun etuna on, että mobiililaitteiden lisäksi saadaan katettua myös tavallisten työpöytäkäyttäjien tarpeet eikä erillisiä sivustoja mobiili- ja työpöytäkäyttöä varten ole tarpeellista ylläpitää. Ylläpito helpottuu myös sovellusta päivittäessä, sillä päivityksiä ei ole tarpeen viedä jokaisen alustan sovelluskauppoihin ladattavaksi, vaan päivitykset ovat käyttäjillä välittömästi käytössä, kun uusi versio on päivitetty sivuston web-palvelimelle.

Käytännössä web-sovellus on lähes kokonaan alustariippumaton, sillä sen toiminta riippuu ainoastaan käytettävästä selaimesta. Selain on myös useimmilla alustoilla mahdollista korvata toisella selaimella, mikäli ongelmia sovelluksen toiminnan kanssa ilmenee. Ongelmana kuitenkin on, että web-sovellusta ei voida asentaa mobiililaitteeseen lukuunottamatta tiettyjä harvemmin käytettyjä alustoja, kuten Tizen [47] tai Firefox OS [42]. Tämä pakottaa käyttäjät spontaanisti vierailemaan sivulla myös jatkossa, sillä erillisiä push-ilmoituksia eli käyttöjärjestelmän tilapalkissa näytettäviä ilmoituksia sovelluksen tilasta sen ollessa suljettuna, ei ole mahdollista antaa, kuten laitteeseen asennettavassa sovelluksessa. Vaihtoehtoisesti web-sovellukselle voidaan tehdä WebView-näkymää hyödyntävä isäntäsovellus, joka on mahdollista asentaa laitteeseen sovelluskaupan kautta. Kyseinen ratkaisu tarkoittaa sitä, että toisesta laitteeseen ajettavasta sovelluksesta yhdistetään web-sovelluksen palvelimeen ja tulkitaan web-sovellusta tämän sisällä WebView-näkymässä, vastaavasti kuin erillisellä selaimella tulkittaisiin. Ongelmana kuitenkin on, että esimerkiksi Apple App Store -sovelluskaupan säännöissä kielletään sovellukset, jotka ainoastaan paketoivat web-sovelluksen sisäänsä [9]. Tämä saattaa aiheuttaa sovelluksen hylkäämisen ja poiston sovelluskaupasta, minkä jälkeen sovellus tulee hyväksyttää kauppaan uudelleen.

WebView-näkymässä ajettavan web-sovelluksen rajoitteena on lisäksi pakollinen verkkoyhteys, sillä sovelluksen ensimmäistä latausta ja useimmiten lähes kaikkia toimintoja suoritetaan ottamalla yhteyttä sovelluksen web-palvelimeen. Käytännössä web-sovelluksen on mahdollista varastoida http-pyyntöjä väliaikaisesti talteen ja suorittaa kutsut vasta

verkkoyhteyden palauduttua. Tässäkin tapauksessa käyttäjällä on kuitenkin oltava verkkoyhteys web-sovellusta avatessaan, jotta sovelluksen päärakenne saadaan ladattua. Tämän vuoksi web-sovellus ei ole oikea vaihtoehto käyttötapauksiin, joissa sovellus nojautuu raskaasti tietosisältöön myös offline-tilassa. [4]

Web-sovelluksen toteuttaminen tuo eteen myös muita verkkosivustoille tyypillisiä haasteita. Sovelluksen suosion kasvattamiseksi täytyy sovelluksen toteutuksessa ja ylläpidossa huomioida hakukonenäkyvyys. Mikäli sivustoa ei löydy hakukoneella tai se häviää muiden hakutulosten joukkoon, on varsinkin uuden sovelluksen vaikea saada käyttäjiä ilman maksettua hakukonemainontaa. Sisällöltään raskaasti JavaScriptiin pohjautuvat web-sovellukset eivät usein ole hakukoneiden mielestä kiinnostavia, sillä niiden hakualgoritmit tarkastelevat ensisijaisesti HTML-sisällön ominaisuuksia. JavaScript-ohjelmistokehyksillä luodut web-sovellukset tuottavat usein suurimman osan HTML-sisällöstään vasta JavaScriptin suorituksen aikana, minkä vuoksi hakukoneet, jotka eivät suorita JavaScript-ohjelmakoodia ryömiessään web-sovellusta läpi, eivät kykene sijoittamaan kyseistä sovellusta ylös tuloksissaan. Huomioitavaa kuitenkin on, että, vastaava ongelma sovelluksen hakusijoituksen kanssa esiintyy myös sovelluskauppojen puolella natiivisovelluksen tapauksessa. Sovelluskauppojen hakutoiminnot perustuvat usein ainoastaan sovelluksen nimeen ja jo saatuun suosioon. Uuden sovelluksen on tämän vuoksi haastavaa saada näkyvyyttä ilman ulkoista mainontaa myös sovelluskaupassa.

2.4 Hybridisovellus

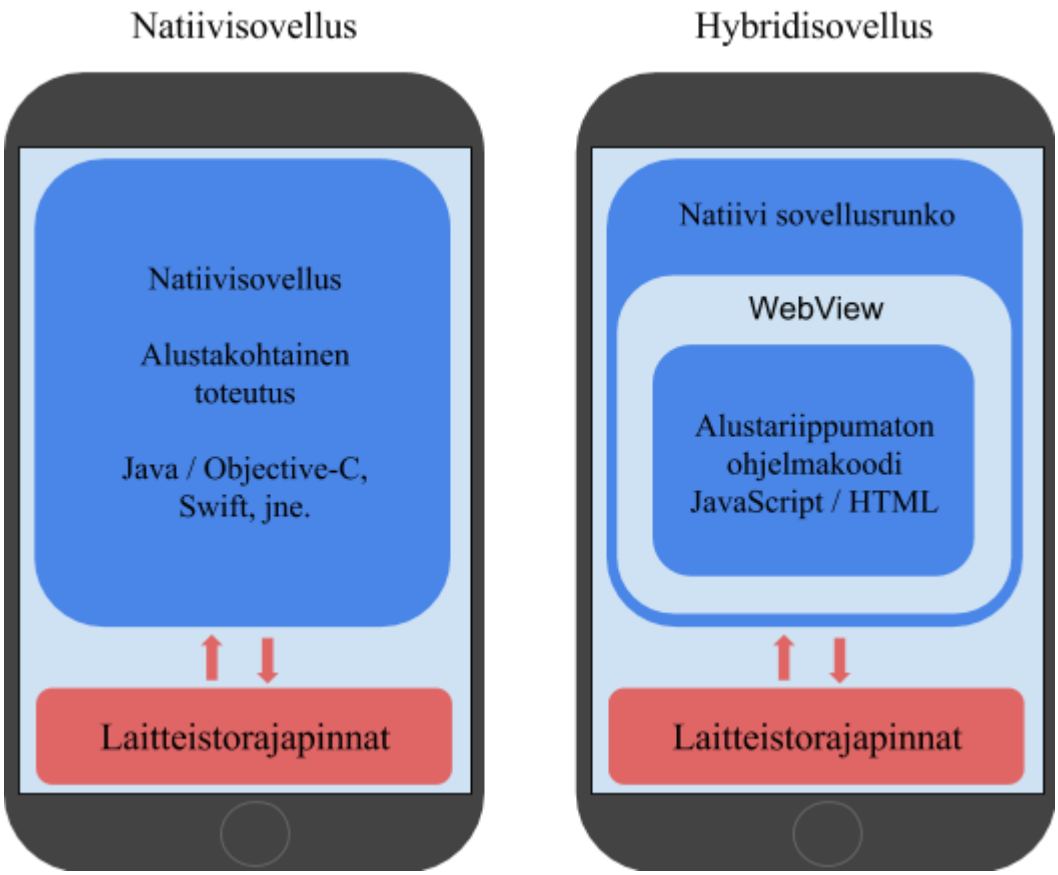
Hybridisovellukset kehitettiin helpottamaan useamman alustan yhtäaikaista sovelluskehitystä mahdollistamalla web-sovellusten kehitykseen käytettyjen teknologioiden hyödyntämisen laitteeseen asennettavien sovellusten kehityksessä. Hybridisovelluksen ideana on, että se voidaan, joko suoraan tai pienellä vaivalla kääntää suoraan useammalle alustalle yhteensopivaksi suoritettavaksi sovellukseksi. Käytännössä hybridisovellus toimii siten, että sovelluksen runko on alustakohtainen natiivisovellus, jonka sisällä suoritetaan alustariippumatonta ohjelmakoodia, esimerkiksi JavaScriptia ja HTML-merkkauskieltä [14].

Hybridisovelluksen kehitys on parhaimmillaan yhtä edullista, kuin yksittäisen natiivisovelluksen kehitys [16]. Haasteena on kuitenkin usein heikompi suorituskyky ja joskus myös huono laitteisto- ja alustayhteensopivuus, mikä saattaa aiheuttaa ongelmia, jos sovellusta ei ole testattu kehityksen alusta asti aktiivisesti jokaisella kohdealustalla. Heikko suorituskyky ilmenee hybridisovelluksesta useimmiten animaatioiden sulavuuden puutteena, sovelluksen pitkänä aukeamisaikana ja suurempana resurssien käyttönä verrattuna natiiviin toteutukseen.

Hybridisovelluksen etuna sovelluksen suosion kannalta on alustariippumattomuuden lisäksi se, että osa ohjelmistokehyksistä, kuten React Native [22], mahdollistavat sovelluk-

sen lähdekoodin uudelleenkäytön myös edellisessä kohdassa käsiteltynä web-sovelluksena [38]. Tämän vuoksi hybridisovellus voidaan kohdistaa myös osittain työpöytäkäyttäjille ja hyötyä sekä web-puolen hakukonenäkyvyydestä, että mobiilialustojen sovelluskauppojen näkyvyydestä. Lisäksi sovelluksen markkinoinnissa voidaan painottaa edullisempaa vaihtoehtoa, joka yleensä on markkinoidusta alueesta riippuen perinteinen web-markkinointi [6]. Hakukoneyhtiöt ovat myös viime aikoina kehittäneet hakutoimintojaan siten, että hakukoneet osaavat jatkossa etsiä myös suoraan sovellusten sisällöstä [31]. HTML-merkkaukseen pohjautuvat hybridisovellukset tulevat todennäköisesti tulevaisuudessa hyötymään tästä, sillä hakualgoritmeja on jo pitkään kehitetty HTML-pohjaisen verkkosisällön analysointiin. Mahdollista kuitenkin on, että algoritmien kehittyessä tämä ero tulee tasoittumaan hybridi- ja natiivisovellusten välillä.

Tyypillistä korkean tason arkkitehtuurieroa natiivi- ja hybridisovellusten välillä käsitellään kuvassa 3. Natiivisovelluksen tapauksessa koko sovellus on toteutettu alustakohtaisesti, kyseisen alustan tukemalla ohjelmointikielellä: Android Javalla [1] ja iOS Objective-C- tai Swift-ohjelmointikielellä [8][11]. Windows Phone alusta puolestaan tukee Visual C++, Visual C# ja Visual Basic -ohjelmointikieliä [40][41]. Poikkeuksia mainittujen kielten tukien lisäksi kuitenkin löytyy, mutta toistaiseksi jokaisen alustan tukemaa yhteistä täysin natiiviksi sovellukseksi käännettävää ohjelmointikieltä ei ole olemassa.



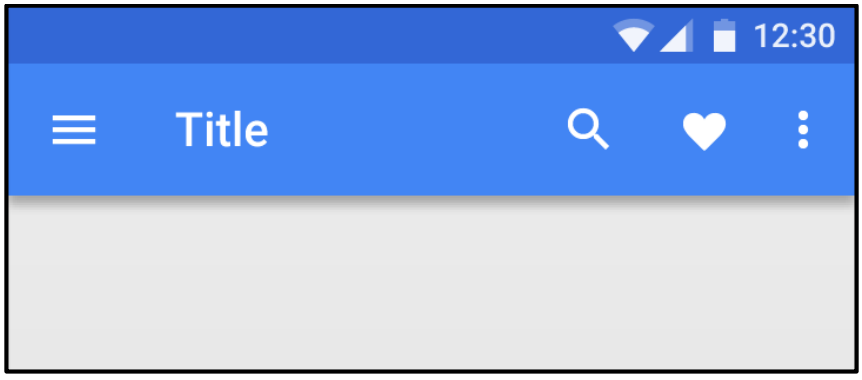
Kuva 3. *Natiivi- ja hybridisovellusarkkitehtuuri*

Hybridisovellukset ratkaisevat yllämainitun ongelman tarjoamalla natiivilla alustakohtaisella ohjelmointikielellä toteutetun valmiin rungon, jonka sisällä voidaan suorittaa alustariippumatonta ohjelmakoodia. Riippuen käytetystä teknologiasta yleensä käyttöliittymä tai suurin osa siitä, sekä sovelluslogiikka voidaan toteuttaa esimerkiksi web-kehityksessä käytetyllä HTML5-merkintäkielellä ja sovelluslogiikka JavaScript-ohjelmointikielellä.

Käyttöliittymän ja sovelluslogiikan lisäksi olennaista hybridisovelluksen toiminnassa on natiivin sovellusrungon ja laitteistorajapintojen välinen kommunikointi. Käytettävän sovellusrungon tulee suoriutua kommunikoinnista laitteistorajapinnan kanssa usealla eri alustalla. Kirjoitushetkellä suosituin JavaScript sovelluslogiikan mahdollistava ohjelmistokehys on Apache Cordova [5], joka tarjoaa integraation laitteistorajapintaan kohdassa 2.1 mainittujen suosituimpien käyttöjärjestelmien lisäksi muun muassa BlackBerry-, Bada- ja Symbian-käyttöjärjestelmille [2]. Cordovan lisäksi kuullaan usein puhuttavan PhoneGap-ohjelmistokehyksestä samassa yhteydessä, toisinaan jopa sekoittaen nimet keskenään. Cordova eriytettiin PhoneGap-ohjelmistokehyksestä vuonna 2012, avoimen lähdekoodin kehitykseen ja kumpikin ohjelmistokehys pohjautuu samaan lähdekoodiin. PhoneGap jatkoi samalla Adoben omistuksessa suljetun lähdekoodin kehityksessä. [39]

2.5 Toteutustavan vaikutus sovelluksen toimintaan

Natiivisovellusten esiinnyttyä markkinoilla alustariippumattomia ratkaisuja pidempään, ovat käyttäjät tottuneet natiivisovelluksen tarjoamaan suorituskykyyn ja ulkoasullisiin ominaisuuksiin. Natiiville sovellukselle on tyypillistä, että tietyt komponentit, kuten latausikonit, valikkopalkit, valitsimet ja painikkeet ovat suurimmalla osalla alustan sovelluksista täysin yhteneviä sekä ulkoasultaan, että toiminnaltaan. Kuvissa 4 ja 5 esitellään Googlen ja Applen tyyliohjeiden mukaiset kullekin alustalle tyypilliset valikkopalkit, joita alustariippumattomat ohjelmistokehykset pyrkivät mukailemaan. [10][28]



Kuva 4. *Googlen material design -tyyliohjetta noudattava valikkopalkki*

Kuva 5. *Applen tyyliohjetta noudattava valikkopalkki*

Näkyvien käyttöliittymäkomponenttien lisäksi olennaista natiivin vaikutelman säilyttämiseksi ovat erilaiset sovelluksen tilojen välillä siirtymistä mallintavat animaatiot. Natiivisovelluksen animaatiot päivitetään 60 ruudun sekuntinopeudella, mikä tekee animaatiosta sulavan. Päivittämättä jätetyt ruudut animaatiossa tekevät sovelluksen käyttämisestä kankeaa ja saattavat rikkoa vaikutelman natiivista sovelluksesta.

Pyyhkäisyeleet, kuten valikon sulkeminen ja avaaminen näytön reunasta pyyhkäisemällä, ovat myös ominaisia natiivisovellukselle. Pyyhkäisyeleiden puuttuessa käyttäjät saattavat toistuvasti yrittää laukaista sovelluksen toimintoa pyyhkäisyllä, mikä huonontaa sovelluksen laatuvaikutelmaa. Pyyhkäisyeleiden lisäksi on olennaista, että jokaiselle käyttäjän syötteelle annetaan palaute. Natiivisovelluksissa tämä toteutetaan usein muuttamalla kosketetun elementin väriä, suorittamalla lyhyt animaatio tai esimerkiksi antamalla käyttäjälle taktiilinen eli tuntoaistiin perustuva palaute laitteen värinämoottoria hyödyntämällä.

Alustasta riippuen natiiville sovellukselle tyypillisiä ominaisuuksia saattaa olla myös enemmän. Yllä mainitut ominaisuudet ovat kuitenkin yleisimpiä käyttäjien oppimia ominaisuuksia Android- ja iOS-alustoilla.

2.6 Kehityskustannukset

Merkittävä syy alustariippumattomien työkalujen nopeaan kehitykseen viime vuosina on niiden kustannustehokkuus verrattuna alustakohtaisesti toteutettaviin ratkaisuihin. Kuuluisa sanonta ”write once, run anywhere” [25] kiteyttää alustariippumattomien teknologioiden lopullisen tavoitteen ja selittää pääsääntöisesti myös kustannuserot. Usein ei kuitenkaan ole täysin mahdollista saavuttaa tätä tilaa, vaan monesti kullekin alustalle joudutaan kirjoittamaan sama osa ohjelmaa hieman eri tavoin. Taulukossa 1 vertaillaan toteutustapojen kustannusrakennetta Comentum Corporationin suorittaman tutkimuksen pohjalta [16]. Tutkimuksessa ei ole huomioitu esimerkiksi käytettävyystutkimuksia, käyttöliittymäsuunnittelua ja projektin hallintaa osaksi kustannuksia, mikä saattaisi pienentää kustannuseroa toteutustapojen välillä, niiden uudelleenkäytettävyyden vuoksi.

Taulukko 1. *Natiivin ja alustariippumattoman toteutuksen kustannukset*

Kehityksen osa-alue	Pienen tai keskisuuren natiivisovelluksen kehityskustannukset	Pienen tai keskisuuren alustariippuman sovelluksen kehityskustannukset
Back-end, rajapinnat, sovelluksen levitys	$10 000 - $20 000	$10 000 - $20 000
Natiivi iOS -toteutus	$15 000 - $30 000	$0
Natiivi Android -toteutus	$10 000 - $20 000	$0
Alustariippumaton toteutus	$0	$10 000 - $20 000
Yhteensä	$35 000 - $70 000	$20 000 - $40 000

Käytännössä alustariippumatonta sovellusta kehittäessä kustannuksia ei suoraan voi jakaa alustojen määrällä verrattuna natiivin kehitysmallin kokonaiskustannuksiin. Usein valittu alustariippumaton kehitystapa aiheuttaa jonkin verran ylimääräisiä kustannuksia kutakin toteutettavaa alustaa kohden. Tämä aiheutuu siitä, että alustariippumatonkaan sovellus ei usein ole ohjelmakoodiltaan kokonaisuudessaan alustariippumaton, jolloin osa sovelluksesta joudutaan kirjoittamaan alustakohtaisesti [51]. Kehitysmallien välillä on kuitenkin jo kahdelle alustalle kehittäessä niin suuri ero, että alustariippumattoman teknologian valinta toteutustavaksi on houkutteleva, mikäli alustariippumaton sovellus vain täyttää asetetut suorituskykykriteerit. Mikäli sovellusta lisäksi alkuperäisen kehitysvaiheen lisäksi ylläpidetään tai jatkokehitetään, kasvavat kustannuserot entisestään alustariippumattoman sovelluksen eduksi varsinaisen ohjelmakoodin tuottamiseen käytetyn pääoman osuuden kasvaessa yhä suuremmaksi.

Formotus Incorporated:n toteuttaman tutkimuksen mukaan ylläpitovaiheessa olevista sovelluksista 29,6 % päivitetään vähintään kerran kuukaudessa ja 52,8 % vähintään kerran puolessa vuodessa. Saman tutkimuksen mukaan kuukausittaiset juoksevat ylläpito- ja jatkokehityskulut ovat keskisuurelle sovellukselle $5000–11 000 ja pienelle sovellukselle noin $3000. Kyseiset kulut koostuvat projektinhallinnasta, suunnittelusta, ominaisuuksien kehittämisestä ohjelmakoodin tasolla, testauksesta, sovelluksen päivittämisestä sovelluskauppoihin ja palvelimille sekä ylläpidollisista toimista. Kehitystavan valinta natiivin ja alustariippumattoman kehitysmallin välillä vaikuttaa mainituista osa-alueista välittömästi ainoastaan uusien ominaisuuksien kehittämiseen. Tutkimuksessa ei eritellä kustannusten tarkkaa jakautumista osa-alueiden välille, sillä ne saattavat vaihdella riippuen kehitetystä sovelluksesta.

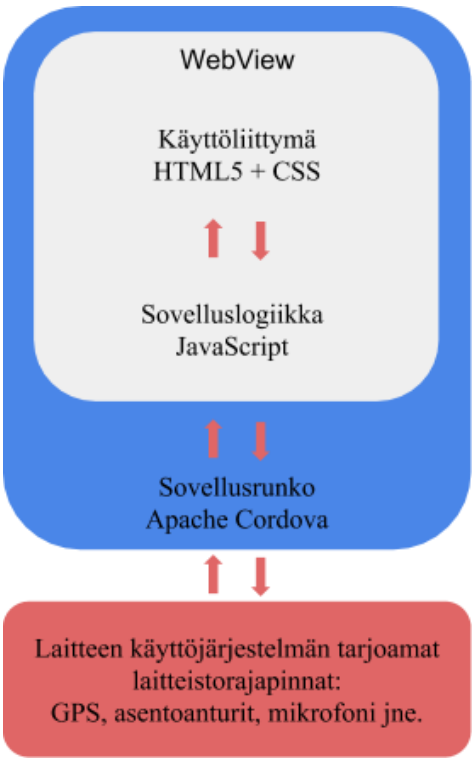
Jos sovellukseen kehitetään aktiivisesti uusia ominaisuuksia, ovat näihin kohdistuvat kustannukset luonnollisesti suuremmat, kuin sovelluksella, jota ainoastaan ylläpidetään päivittämällä nykyisten ominaisuuksien yhteensopivuutta. Alustariippumaton kehitysmalli on tästä syystä kustannustensa puolesta kannattava kehitystapa erityisesti tilanteissa, joissa sovellukseen kehitetään runsaasti ominaisuuksia myös ylläpitovaiheessa. [24].

3. TUTKITTAVAT SOVELLUSTEKNIIKAT

Tässä luvussa käydään läpi työssä tutkittavien teknologioiden perusperiaatteet ja arkkitehtuurit. Tutkittaviksi sovellustekniikoiksi valittiin Ionic ja React Native -ohjelmistokehykset, joilla pyritään toteuttamaan sovellus, jota tavallisen käyttäjäkunnan edustaja ei kykene erottamaan natiivista sovelluksesta. Kunkin toteutustekniikan ohessa käsitellään myös toteutuskeinot, joilla kehittäjä voi vaikuttaa sovelluksen suorituskykyyn.

3.1 Ionic

Ionic on Drifty Co:n kehittämä avoimen lähdekoodin ohjelmistokehys, joka tarjoaa kokoelman valmiita mobiililaitteille räätälöityjä HTML5-käyttöliittymäelementtejä sekä mahdollisuuden kirjoittaa sovelluksen logiikka alustariippumattomasti JavaScriptilla [18]. Ionicilla kirjoitettu sovellus paketoidaan laitteessa suorittamista varten natiivisovellukseksi hyödyntäen Apache Cordovaa, ja sovelluksen näkymiä suoritetaan WebView:ksi kutsutun natiivisovelluksen tarjoaman selainta emuloivan näkymän sisällä. Apache Cordovan ja Ionicin muodostama arkkitehtuuri on annettu kuvassa 6.



Kuva 6. *Apache Cordova ja Ionic -arkkitehtuuri*

Käytännössä Apache Cordova mahdollistaa sovelluksen toteuttamisen lähes millä tahansa JavaScript-kehyksellä, joten toteutus ei abstraktiotasoltaan välttämättä eroa millään tavoin tavallisen JavaScriptia käyttävän verkkosivuston kehityksestä. Sovelluskehityksen

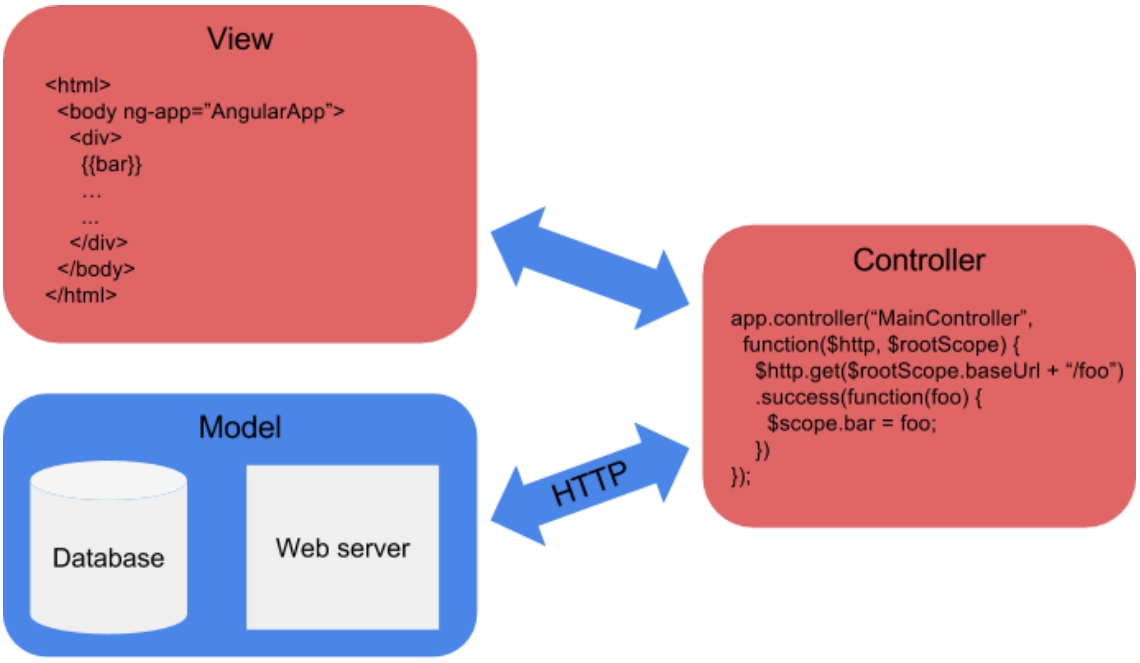
kannalta Ionicin etuna on ennen kaikkea tavallisten web-kehityksessä käytettyjen tekniikoiden hyödyntäminen. Tämä on eduksi erityisesti tilanteissa, joissa ohjelmistokehittäjillä ei ole kokemusta sovellusten kehittämisestä mobiilialustoille. Koska Ionic hyödyntää tavallisessa web-kehityksessä suuren suosion saavuttanutta AngularJS-ohjelmistokehystä, pystytään Ionic-sovellusta kehittäessä hyödyntämään web-kehittäjillä jo olemassaolevaa tietotaitoa. [18][26]

Teknisestä toteutustavastaan johtuen Ionic ei tarjoa natiiveja käyttöliittymäkomponentteja React Nativen -ohjelmistokehyksen tavoin. Tämän sijaan Ionic tarjoaa kokoelman mukautettuja HTML elementtejä, CSS-muotoiluja ja fontteja. Suurin osa Ionic-kehyksen tarjoamista elementeistä korvataan tai niitä täydennetään yhdellä tai useammalla HTML5-standardin mukaisella elementillä, ennen kuin ohjelma mallinnetaan WebView-näkymässä. Esimerkiksi *ion-content*-elementin alle luodaan elementti * *, joka mahdollistaa näkymän vierittämisen pystysuunnassa, mikäli kyseisen elementin sisällä olevat elementit eivät mahdu laitteen ruudulle samanaikaisesti. Osa elementeistä myös sisältää alustan mukaan vaihtuvaa toiminnallisuutta. Esimerkiksi ion-spinner-elementti korvataan alustakohtaisella natiivia latausikonia muistuttavalla SVG-ikonilla.

3.2 Toteutuksen vaikutus Ionic-sovelluksen suorituskykyyn

Ionic-sovelluksen suorituskyky riippuu pitkälti tämän hyödyntämästä AngularJS-kehyksestä, minkä vuoksi Ionic-sovelluksen suorituskykyä optimoidakseen täytyy kehittäjän ymmärtää ensin jonkin verran myös AngularJS:n periaatteista. AngularJS on Googlen vuonna 2010 julkaisema JavaScript-kehys, joka on saavuttanut suuren suosion web-kehityksessä tarjoamalla mahdollisuuden rakentaa web-sovelluksen MVC-arkkitehtuurin mukaisesti [26]. AngularJS:n pääperiaatteena on, että sivusto tai web-sovellus rakennetaan single page application -periaatteella, eli sovellus koostuu päänäkymästä, jonka päälle ladataan dynaamisesti uutta dataa ja muita osanäkymiä. Tämä mahdollistaa sen, että sivua ei ladata missään vaiheessa uudelleen, mikä luo käyttäjälle enemmän mielikuvan sovelluksesta, kuin perinteisestä verkkosivustosta. [49]

AngularJS-sovelluksen perusrakenne koostuu näkymistä (view), kontrollereista (controller) sekä usein esimerkiksi REST-rajapinnasta, joka toimii sovelluksen mallina (model). Mainittujen komponenttien suhdetta sovelluksessa käsitellään kuvassa 7. Kuvasta poiketen on käytäntönä useimmiten myös AngularJS:n palveluiden (service) käyttö kontrollerin ja mallin välissä, mikä mahdollistaa helpommin hallittavan ja uudelleenkäytettävän komponenttijaon. Palveluiden tarkoituksena on paketoida AngularJS-sovelluksen raskaampi toiminnallisuus oman abstraktionsa taakse. Palvelut ovat singleton-luokkia, joten niistä on mahdollista luoda vain yksi instanssi. Kyseinen instanssi voidaan tämän jälkeen injektoida tarvittaviin kontrollereihin laajentamaan näiden toiminnallisuutta.

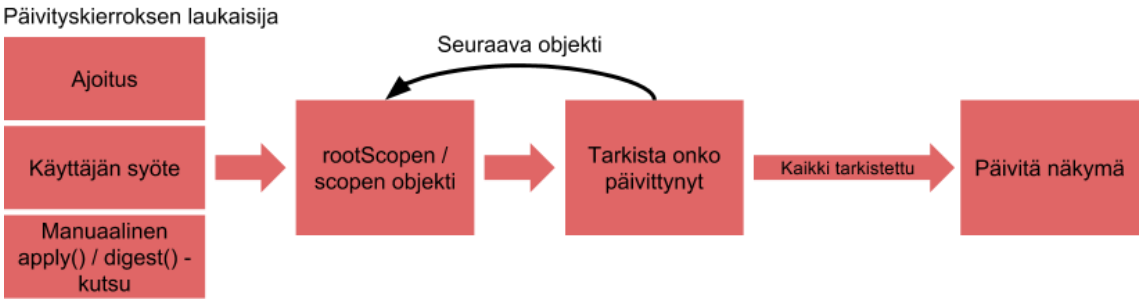


Kuva 7. *AngularJS-sovelluksen MVC-arkkitehtuuri*

Sovelluksen osanäkymien vaihdosta huolehtii reititin (router). AngularJS:n osana käytetään usein ngRoute-nimistä reititintä, joka aiemmin kuului oletuksena viralliseen AngularJS-kirjastoon. Ionic käyttää oletusreitittimen sijaan UI Router -nimistä reititintä, jonka tavoitteena on tarjota ngRouten tarjoamien ominaisuuksien lisäksi myös muita edistyneempiä ominaisuuksia. NgRoutesta poiketen UI Router käsittelee reittejä tiloina (state), jotka eivät pohjaudu yksinomaan sovelluksessa aktiivisena olevaan URL:iin eli reittiin. Tiloille on mahdollista kohdistaa tietty reitti sovelluksessa, mutta tämä ei ole pakollista. Tämä mahdollistaa sen, että yksittäisellä reitillä voi olla useampi tila, minkä mukaan voidaan suorittaa haluttua logiikkaa sovelluksessa. Yksittäinen tila voi myös koostua useammasta osanäkymästä ja näkymät voivat olla sisäkkäisiä. Tämä mahdollistaa sen, että monimutkaiset näkymät voidaan jakaa useampaan pienempään näkymään, joita voidaan myös uusiokäyttää tehokkaammin osana muita näkymiä ja pienentää samalla sovelluksen kokoa.

Sovelluksen pieni koko on luonnollinen etu etenkin web-sovellusta kehittäessä, mutta samoja periaatteita on mahdollista soveltaa myös toteutettaessa hybridisovellusta. Web-sovelluksen tapauksessa etu saadaan pääasiassa lyhyemmästä latausajasta, mikä johtaa useimmiten myös parempaan käyttäjäkokemukseen sekä parempaan hakukonenäkyvyyteen [52][45]. Koska HTML5-pohjaisia hybridisovelluksia ajetaan käytännössä sovelluksen sisäisessä selaimessa, ovat suorituskykyvaikutukset vastaavanlaisia. Sovelluksen tiedostojen suurempi koko saattaa ilmetä esimerkiksi käyttöliittymäelementtien välähtelynä, animaatioiden kankeutena ja pidempinä latausaikoina näkymää vaihtaessa sekä sovellusta käynnistäessä. Tämä saattaa johtaa huonoon käyttäjäkokemukseen, sillä natiivisovelluksia käyttäessä vastaavia suorituskykyongelmia harvoin esiintyy. [3]

AngularJS käyttää tiedon sitomiseen näkymän ja kontrollerin välillä two-way data binding nimistä tekniikkaa [27]. Näkymissä näytettävät muuttujat tallennetaan *$scope*-nimiseen objektiin, jota on mahdollista lukea ja kirjoittaa sekä näkymän, että kontrollerin puolelta. Kontrollerikohtaisen *$scope*-objektin lisäksi AngularJS tarjoaa myös *$rootScope*-nimisen objektin, johon voidaan tallentaa arvoja, joita halutaan käsitellä globaalisti. Kontrollerin puolella tehdyt muutokset sekä *$rootScope*-objektiin, että *$scope*-objektiin näkyvät automaattisesti näkymän puolella ja päinvastoin, tämän vuoksi tekniikkaa kutsutaan kaksisuuntaiseksi. Kaksisuuntaisuus saadaan aikaan tarkkailemalla jatkuvasti muutoksia *$scope*-objektissa. Tämä saadaan aikaan kohdistamalla automaattisesti jokaista *$scope*-objektin sisältämää objektia kohden oma tarkkailija (watcher) AngularJS:n *$watch*-metodilla, kun *$scope*-objektiin lisätään arvo näkymässä. Näkymän päivitykset suoritetaan erillisellä päivityskierroksella (digest cycle), joka kutsuu jokaista olemassaolevaa tarkkailijaa. Päivityskierros alkaa *$rootScope*-objektin tarkkailijoista ja siirtyy tämän jälkeen *$scope*-objektiin. Kukin tarkkailija vertaa nykyistä arvoaan edellisen kierroksen arvoonsa ja päivittää mahdollisen muutoksen *$scope*-objektiin. Päivityskierroksen kulku on esitetty kuvassa 8.



Kuva 8. *Päivityskierroksen kulku*

AngularJS:n datanhallintamalli on tehokas sovellusta toteuttaessa, sillä dataa ei ole tarpeen päivittää tai välittää näkymän ja kontrollerin välillä käsin erillisillä päivityskutsuilla. Sovelluksen suorituskyvyn kannalta kyseinen malli on kuitenkin ongelmallinen, sillä monimutkaisessa sovelluksessa tarkkailijoita saattaa olla niin paljon, että niiden päivittämiseen vaaditaan huomattava määrä laskentatehoa. Ongelma korostuu etenkin mobiililaitteilla, mutta ongelmaan on syytä kiinnittää huomiota myös työpöytäkäytössä varsinkin suurempia tietomääriä mallintaessa. Tarkkailijoiden maksimimäärän ohjearvona pidetään n. 2000 tarkkailijan rajaa, minkä jälkeen suorituskyvyssä on havaittavissa heikentymistä. [43]

Tarkkailijoiden määrää on mahdollista vähentää hyvällä suunnittelulla. Normaalin datan sidonnan sijaan on esimerkiksi mahdollista käyttää one-time binding -ilmaisua, jolloin muuttujaan sidotaan arvo ainoastaan kerran, eikä sille tämän jälkeen luoda tarkkailijaa. Tämä tapa soveltuu tilanteisiin, joissa tiedetään, että data ei muutu käytön aikana. Esimerkki one-time binding -ilmaisun käytöstä löytyy ohjelmasta 1.

```
<div ng-repeat=”foo in bars”>

      <!-- Normal binding -->
      {{foo}}

      <!-- One-time binding -->
      {{::foo}}
</div>
```

Ohjelma 1. *Esimerkki one-time binding -ilmaisun käytöstä.*

Tarkkailijoiden määrää on myös mahdollista vähentää säilyttämällä datan automaattinen sidonta muissa muuttujissa paitsi niissä, joista tarkkailija on poistettu. Tarkkailija voidaan poistaa kutsumalla AngularJS:n *$watch*-metodin palauttamaa metodia. Tämä onnistuu ohjelmassa 2 kuvatulla tavalla.

```
// App.js
angular.module(”app”, [])
.controller(”MainController”, function($scope) {

      $scope.numberOfDigestCycles = 0;

      $scope.removeWatcher = function() {

            // Function returned by $scope.watch() gets called and the watcher
            // is deleted
            sampleWatcher();
      }

      var sampleWatcher = $scope.watch(”modelBeingWatched”,
      function(newValue, oldValue) {
            if(newValue != oldValue) $scope.numberOfDigestCycles++;
      });
});

<!-- index.html -->
<body ng-controller=”MainController”>
      <input ng-model=”modelBeingWatched> {{numberOfDigestCycles}}
      <button ng-click=”removeWatcher()”>Remove watcher</button>
</body>
```

Ohjelma 2. *Esimerkki tarkkailijan manuaalisesta poistosta.*

3.3 Web-teknologiat osana Ionic-ohjelmistokehystä

Toteutustapojen lisäksi suurta roolia suorituskyvyssä näyttelee käytetty selain. Kuten internet-selain, myös Cordova-ohjelmistokehykseen pohjautuvien sovellusten käyttämä WebView-näkymä on kohtalaisen helposti vaihdettavissa. Oletuksena sovellus käyttää aina laitteen omaa WebView-näkymää, jonka tarjoamat rajapinnat ja suorituskyky saattavat vaihdella laitteen käyttöjärjestelmän version mukaan. Käytännössä Android-alustalla WebView vastaa Android-selainta ennen Android-versiota 4.4. Android-versiosta 4.4 alkaen oletuksena käytössä oleva WebView pohjautuu Chromium-selaimeen [37]. IOS-alustalla vastaavasti ennen iOS versiota 8 käytössä on oletuksena UIWebView ja

tästä eteenpäin WKWebView [48]. Sekä Android-, että iOS-alustoilla vanhemmat WebView-näkymät ovat suorituskyvyltään selkeästi heikompia kuin uudet. Sovellusta toteuttaessa tulee tämän vuoksi arvioida Android 4.4- ja iOS 8-versioita vanhempien käyttöjärjestelmien osuus sovelluksen loppukäyttäjillä. Suorituskykyerot ovat huomattavia, sillä esimerkiksi iOS:n UIWebView ja WKWebView-näkymien suorituskyvyssä on n. 20 % suorituskykyero sivun mallinnuksessa ja jopa yli 300 % parannus pelkän JavaScript-ohjelmakoodin suorituksessa WKWebView:n eduksi. [48][44]

Suurin osa vanhempien alustojen WebView-näkymien suorituskykyongelmista voidaan korjata ottamalla käyttöön esimerkiksi Intelin avoimen lähdekoodin Crosswalk-WebView. Crosswalkin etuna on paremman suorituskyvyn lisäksi se, että WebView pohjautuu jokaisella alustalla samaan selaimeen, jolloin kehitettävän sovelluksen testaus helpottuu. Crosswalk lupaa lisäksi ruudunpäivitysnopeuden olevan 20-60 ruutua sekunnissa alustan oletuksena käyttämän WebView:n tarjotessa yleensä n. 5-10 ruudun päivitysnopeutta sekunnissa. [23]

Kuten web-sovelluksissa, myös Ionic-sovelluksen JavaScript-, CSS- ja HTML-tiedostot on mahdollista minifioida ja niputtaa yhteen. Tämä tarkoittaa, että samantyyppiset tiedostot yhdistetään ja tiedostoista poistetaan kaikki ylimääräiset merkit, säilyttäen samalla alkuperäinen toiminnallisuus. Kyseinen prosessi tekee ohjelmakoodista vaikealukuista ihmiselle, mutta tiedoston koon pienentyessä se on nopeampi siirtää ja lukea koneellisesti. Ionic-sovelluksessa toimenpide parantaa suorituskykyä etenkin näkymää vaihtaessa ja sovellusta avatessa, kun uutta ohjelmakoodia joudutaan lukemaan laitteen muistista WebView-näkymän muistiin. [3]

Mikäli sovelluksen näkymiä tai muita sovelluksen resursseja ladataan verkon ylitse, on olennaista huomioida myös HTTP/1.1-standardin rajoitteet. Koska HTTP/1.1 kykenee ainoastaan 2-8 yhtäaikaiseen yhteyteen domainia kohden riippuen käytetystä internet-selaimesta, joutuu useimmiten osa ladattavista resursseista odottamaan muiden resurssien latauksen valmistumista. Tämän vuoksi sovelluksen suorituskyvyn kannalta on olennaista optimoida tiedostojen koko ja määrä tai jakaa ne ladattavaksi useamman domainin kautta, esimerkiksi hyödyntämällä sisällönjakeluverkkoa. HTTP/2-standardin yleistyessä kyseiset ongelmat kuitenkin vähenevät, sillä samanaikaisten yhteyksien määrää ei enää ole standardin osalta rajoitettu vaan rajoite asetetaan web-palvelimen toimesta. HTTP/2-standardin server push -ominaisuus parantaa myös suorituskykyä, sillä web-palvelin kykenee tarjoamaan resursseja asiakkaalle jo ensimmäisen pyynnön yhteydessä ilman, että kyseisiä resursseja pyydetään erikseen. [7]

Shadow- ja opacity-ominaisuuksien käyttö Ionic-sovelluksen CSS-tyyleissä, aiheuttaa useimmilla mobiililaitteilla ruudunpäivitysnopeuden pienenemistä. Ongelma esiintyy lähinnä animaatioiden yhteydessä tai jos käyttöliittymän elementit muusta syystä liikkuvat, esimerkiksi vierittämisen yhteydessä. Ongelma on korjattavissa käyttämällä CSS-tyylien sijaan kuvia, jotka ovat osittain läpinäkyviä tai varjo on piirretty kuvaan valmiiksi. [46]

3.4 React Native

React Native on Facebookin kehittämään React-JavaScript-ohjelmistokehykseen [21] pohjautuva alustariippumaton ohjelmistokehys, jonka avulla on mahdollista kehittää käyttöliittymältään natiiveja sovelluksia [22]. Alustariippumaton osa React Nativella kirjoitetusta sovelluksesta on sovelluksen logiikka, joka toteutetaan käyttäen JavaScriptin syntaksia laajentavaa JSX-syntaksia, joka käännetään puhtaaksi JavaScriptiksi ennen ohjelman suoritusta.

React Native -sovellus on mahdollista kirjoittaa yhdistäen JavaScript-ohjelmakoodissa ES5-, ES6- ja ES7-standardeja. ES6-standardi mahdollistaa luokkien kirjoittamisen, mikä ohjaa suunnittelemaan sovelluksen rakenteen komponenttipohjaisesti. Käytännössä React Native -sovellus koostuu useista luokista, jotka voivat esittää kokonaista näkymää tai esimerkiksi nappia. Ohjelmassa 3 käsitellään näkymänä toimivaa *HomeScreen*-luokkaa sekä tämän sisältämää *Button*-luokkaa ES6-standardin mukaisesti kirjoitettuna.

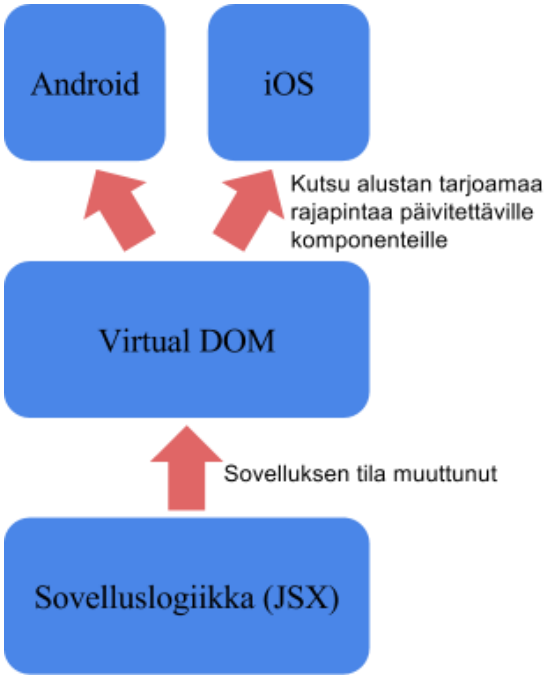
```
class HomeScreen extends Component {
  render() {
    return (
      <Button
        onPress={this.onButtonPress.bind(this)}
        buttonText="Submit">
      </Button>
    );
  }
}
```

Ohjelma 3. *Button-komponentti näkymän sisällä.*

Tavallisesti web-kehityksessä käytettävät JavaScript-kehykset tarjoavat käyttäjän käyttämälle selaimelle käyttöliittymän HTML-merkkauksella kirjoitettuna dokumenttioliomallina. Dokumenttioliomalli on puumainen rakenne valittua merkkauskieltä, jonka selain kykenee piirtämään ihmisen ymmärtämään visuaaliseen muotoon. React poikkeaa tästä tarjoamalla sovelluslogiikan ja käyttöliittymän väliin Virtual DOM:ksi kutsutun mallinnuksen JSX-syntaksilla kirjoitetuista käyttöliittymäkomponenteista. Kyseinen rakenne tarjoaa edun suorituskyvyssä, sillä Virtual DOM voidaan pitää jatkuvasti muistissa ja päivittää käyttäjälle näkyvään dokumenttioliomalliin ainoastaan muuttunut sisältö. Suorituskykyoptimointi ei kuitenkaan ole pääsyy siihen miksi kyseistä rakennetta pidetään mullistavana. Virtual DOM mahdollistaa sen, että tavallisen selaimen ymmärtämän dokumenttiolimallin sijaan voidaan käyttöliittymä mallintaa käytännössä mihin tahansa muotoon. Tämä tarkoittaa kehittäjän kannalta käytännössä sitä, että React-ohjelmistokehyksen hallitsemalla on mahdollista toteuttaa web- ja mobiilisovellusten lisäksi muillekin alustoille, joita ei vielä välttämättä ole edes keksitty.

React Native hyödyntää Reactin Virtual DOM ominaisuutta pyytämällä alustan tarjoamaa käyttöliittymärajapintaa piirtämään halutut natiivit komponentit käyttöliittymään Virtual

DOM:n tilan perusteella. Tämän vuoksi React Native ei tarvitse muista alustariippumattomista JavaScript-kehyksistä tuttua WebView-näkymää käyttöliittymän esittämiseen. Kuvassa 9 esitellään käyttöliittymän päivityksen kulku Virtual DOM:n kautta React Native -sovelluksessa.



Kuva 9. *React Native-sovelluksen käyttöliittymän mallinnussykli.*

Kirjoitushetkellä React Native tarjoaa Android- ja iOS-alustoille yhteisesti taulukossa 2 esitellyt käyttöliittymäkomponentit. Taulukossa vasemmalla on käyttöliittymäkomponentin nimi, joka useimmiten vastaa alustoilla natiivina tarjolla olevaa käyttöliittymäkomponenttia. Vasemmalla taulukossa on lyhyt kuvaus kyseisen komponentin roolista ja toiminnasta.

Taulukko 2. *React Nativen tarjoamat käyttöliittymäkomponentit*

Käyttöliittymäkomponentin nimi	**Kuvaus**
Image	Kuva
ListView	Lista
MapView	Karttanäkymä
Modal	Päänäkymän päälle aukeava popup-tyyppinen näkymä
Navigator	Navigaatio, sisältää myös navigoinnin pyyhkäisyeleillä
Picker	Komponentti yhden vaihtoehdon valintaan useammasta vaihtoehdosta
RefreshControl	Sivupäivityksen hallinta
ScrollView	Vieritettävä näkymä
Slider	Liukupalkki
StatusBar	Sovelluksen tilapalkki
Switch	Päälle/pois -valinta
Text	Tekstin esitysalue

TextInput	Muokattava tekstikenttä
TouchableHighlight	Sisälle asetettu komponentti korostetaan koskettaessa
TouchableOpacity	Sisälle asetetun komponentin läpinäkyvyyttä lisätään koskettaessa
TouchableWithout-Feedback	Kosketettava komponentti, joka ei anna palautetta
View	Sivupohjakomponentti, joka määrittää näkymän pohjatyylin. Voi sisältää muita komponentteja
WebView	Sovelluksen sisälle avautuva selainnäkymä

Lisäksi kehys tarjoaa osalle käyttöliittymäkomponenteista myös alustakohtaiset toteutukset, jotka on esitelty taulukossa 3. Alustakohtaiset komponentit tarjoavat kyseiselle alustalle ominaisia ominaisuuksia, kuten TabBarIOS, joka tarjoaa iOS-alustalle ominaisen välilehtinavigaation ruudun alalaitaan.

Taulukko 3. *React Nativen tarjoamat alustakohtaiset käyttöliittymäkomponentit*

Käyttöliittymäkomponentin nimi	**Alusta**	**Kuvaus**
DrawerLayoutAndroid	Android	Oletuksena piilossa oleva päänäkymän ulkopuolelta näkyviin pyyhkäistävä valikko
ProgressBarAndroid	Android	Latauspalkki
PullToRefreshViewAndroid	Android	Näkymän yläreunan yli vierittäessä laukaistaan näkymän päivitys
ToolbarAndroid	Android	Yläreunan työkalupalkki
TouchableNativeFeedback	Android	Alustalle tyypillinen palaute kosketukseen
ViewPagerAndroid	Android	Sivutettu joukko näkymiä, joiden välillä mahdollista siirtyä pyyhkäisemällä vaakasuunnassa
ActivityIndicatorIOS	iOS	Latauskuvake
DatePickerIOS	iOS	Päivämäärän/ajan valinta
NavigatorIOS	iOS	Navigaatio, sisältää myös iOS-alustan tyypilliset pyyhkäisyeleet
PickerIOS	iOS	Valinta listasta vaihtoehtoja
ProgressViewIOS	iOS	Latauspalkki
SegmentedControlIOS	iOS	Vaakasuuntainen valintakomponentti
SliderIOS	iOS	Liukupalkki
TabBarIOS	iOS	Näkymän alareunassa sijaitseva välilehtivalitsin
TabBarIOS.Item	iOS	Välilehtivalitsimen välilehti

Yllä mainittuja komponentteja ei ole mahdollista laajentaa kehittäjän toimesta, mutta uusia komponentteja on mahdollista toteuttaa. Vaihtoehtoisesti uusi mukautettu komponentti voidaan kuitenkin paketoida jonkin olemassa olevan komponentin sisään ohjelmassa 4 esitetyllä tavalla. Esimerkkiohjelmassa luodaan uusi nappia esittävä komponentti Button, joka sisältää *TouchableHighlight*- sekä *Text*-komponentit. *TouchableHighlight* lisää napille kosketukseen reagoivia ominaisuuksia, esimerkkiohjelman tapauksessa painalluksen yhteydessä vaihtuvan värin. Napille on myös mahdollista liittää muuta toiminnallisuutta, esimerkiksi metodi, jota kutsutaan nappia painaessa. Napin sisältämä *Text*-komponentti puolestaan vastaanottaa *buttonText*-ominaisuutensa kautta napin sisällä esitettävän tekstin.

```
class Button extends Component {
  render() {
    return (
      <TouchableHighlight
        style={styles.button}
        onPress={this.props.onPress}
        underlayColor='#1b6698'>
        <Text style={styles.buttonText}>
          {this.props.buttonText}
        </Text>
      </TouchableHighlight>
    );
  }
};
```

Ohjelma 4. *Käyttöliittymäkomponentin paketointi olemassaolevan React Native käyttöliittymäkomponentin sisään.*

React Native -sovelluksen suorittamiseksi laitteessa, tulee sovellus paketoida alustakohtaisesti, joko Android- tai iOS-alustalle. Android-alustalla tämä onnistuu esimerkiksi Android Studio -kehitysympäristön avulla, ja iOS-alustalla vaaditaan vastaavasti Xcode-kehitysympäristö, jonka käyttöä varten tarvitaan OS X -käyttöjärjestelmää käyttävä laite. Kumpaankin kehitysympäristöön on sisäänrakennettu työkalut sovelluksen paketoimiseen asennuspaketiksi, sekä sovelluksen asentamiseen suoraan saatavilla olevaan laitteeseen. [19]

3.5 Toteutustavan vaikutus React Native -sovellukseen

React Nativen tarjoamat poikkeavat ratkaisut kehyksen arkkitehtuurissa tarjoavat selkeän edun sovelluksen käyttöliittymän suorituskykyyn. Koska käyttöliittymäkomponentit on toteutettu natiivisti WebView-näkymässä tulkittujen HTML-elementtien sijaan, kyetään käyttöliittymää päivittämään 60 ruudun sekuntinopeudella, kuten täysin natiivissa sovelluksessa. Vertailun vuoksi HTML5-pohjaiset WebView-näkymää hyödyntävät sovellukset kykenevät yleensä 5-10 ruudun päivitysnopeuteen. Vaihtoehtoisesti erillistä kolmannen osapuolen WebView-näkymää hyödyntämällä on mahdollista saavuttaa 20, 30 tai paikoitellen jopa 60 ruudun päivitysnopeuksia [23][33].

Suuren ruudunpäivitysnopeuden ylläpito vaatii kuitenkin huolellista suunnittelua myös kehittäjältä. 60 ruudun päivitysnopeus sekunnissa tarkoittaa samalla, että sovelluslogiikalle jää 16,67ms aikaa ladata tarvittavat komponentit käyttöliittymään ennen kuin sovellus joutuu pudottamaan ruudunpäivitysnopeuttaan. Tämä johtuu siitä, että sovelluksen komponenttien animaatiot ja sovelluslogiikka ajetaan samassa JavaScript-säikeessä. Koska käyttöliittymän komponentit päivitetään aina JavaScript-säikeen tapahtumasilmukan lopuksi, säikeen suorituksen viivästyessä myös komponenttien päivityskutsut viivästyvät. Jos siis sovelluslogiikalla kestää esimerkiksi 300ms rajapintakutsun valmistumista odottaessa, joudutaan hylkäämään 18 päivitettävää uutta ruutua, mikä näkyy käyttäjälle animaatioiden kankeutena, ja sovellus ei tänä aikana myöskään vastaa syötteisiin. Osa

komponenteista, kuten NavigatorIOS, poikkeaa tästä suunnittelumallista, sillä ne suoritetaan omassa säikeessään erillään sovelluslogiikan JavaScript-säikeestä. Tämä mahdollistaa sen, että kyseisten komponenttien animaatioita voidaan suorittaa keskeytyksettä rinnakkain sovelluslogiikan kanssa. NavigatorIOS on nimensä mukaisesti yhteensopiva ainoastaan iOS-alustan kanssa, minkä vuoksi useimmissa tapauksissa on suositeltavaa käyttää Navigator-komponenttia paremman alustariippumattomuuden takaamiseksi. Navigator-komponenttia käyttäessä kehittäjän tulee kiinnittää huomiota sovelluslogiikan toteutustapaan, jotta käyttöliittymän päivitystä ei keskeytetä kriittisellä hetkellä. Käytännössä toteutuksessa on usein mahdollista käyttää käyttöliittymän päivityssyklin huomioivia komponentteja, kuten ListView-listakomponentti. Muissa tapauksissa ohjelmakoodi on kirjoitettava niin, ettei ohjelman suoritus pysähdy kerralla odottamaan 16,67ms ajanjaksoa pidempään. [22]

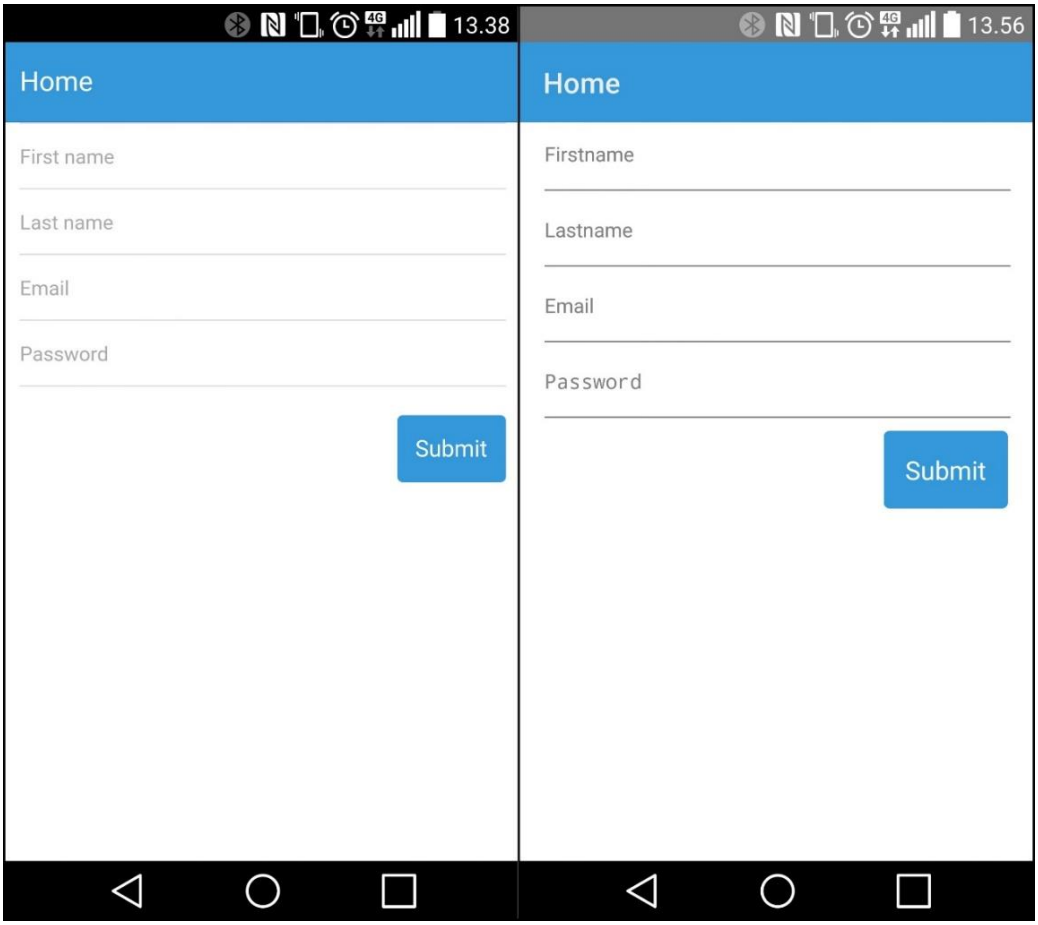
Poiketen kohdassa 3.1 käsitellystä Ionic -ohjelmistokehyksestä React Nativen arkkitehtuurissa tieto kulkee vain yhteen suuntaan. Tämä ratkaisee myös aiemmin käsitellyt suorituskykyongelmat, sillä AngularJS:lle tyypillisille tarkkailijoille ei ole tarvetta. Tarvittaessa vastaavanlainen two-way data binding -mallia noudattava toteutus on mahdollinen hyödyntämällä ReactLink-moduulia. Kyseisen moduulin runsaaseen käyttöön liittyy kuitenkin vastaavat suorituskykyongelmat, kuin AngularJS:n two-way data binding -mallin kanssa, joten sitä tulee käyttää harkiten.

4. MITTAUSTEN TOTEUTUS

Tässä luvussa käydään läpi työssä toteutetut testisovellukset ja niille mittauksia varten asetetut kriteerit. Mittauksia varten toteutettiin kaksi kappaletta Android-alustalla suoritettavia testisovelluksia. Ensimmäinen sovelluksista on WebView-tekniikkaa edustava Ionic-sovellus ja toinen käyttöliittymältään natiivia sovellusta edustava React Native -sovellus.

4.1 Testisovellukset

Työssä toteutettiin kaksi testisovellusta, joiden suorituskykyä optimoitiin luvussa 3 käsitellyin toteutuskeinoin. Ionic-testisovellus toteutettiin ohjelmistokehyksen versiolla 1.2.4 ja React Native -sovellus versiolla 0.17.0. Testisovellukset koostuvat kolmesta näkymästä, joiden sisällöt edustavat tavallisessa tietopohjaisessa sovelluksessa usein käytettäviä komponentteja. Kuvassa 10 nähdään testisovellusten ensimmäinen näkymä, jossa on mahdollista täyttää lomake ja lähettää se. Kuvassa vasemmalla on Ionic- ja oikealla React Native -toteutus.



Kuva 10. *Ionic- ja React Native -testisovellusten ensimmäinen testinäkymä*

Ionic-kehyksen tapauksessa näkymä koostuu HTML-elementeistä ja Ionicin tarjoamista CSS-muotoiluista. Ionic-kehyksen CSS-muotoilut käännetään SASS-tyylittelykielellä kirjoitetusta tyylikirjastosta, johon on testisovelluksen tapauksessa kirjoitettu mukautettuja tyylejä. Ohjelmassa 5 esitellään Ionic-testisovelluksen päänäkymän HTML-kuvaus. Ohjelmasta nähdään AngularJS:n tarjoama two-way data binding -ominaisuus, joka on suorituskykyongelmistaan huolimatta hyödyllinen erityisesti lomakkeiden tapauksessa. Lomakkeen tekstikentän arvon muuttuessa ei ole tarpeen erikseen päivittää uutta arvoa, sillä arvon säilyttävä muuttuja sidotaan suoraan tekstikenttään ng-model-attribuutilla.

```
<ion-header-bar class="bar-positive">
    <h1 class="title">Home</h1>
</ion-header-bar>
<ion-content>
    <div class="list">
        <label class="item item-input">
            <input ng-model="form.firstName"
                    type="text"
                    placeholder="First name">
        </label>
        <label class="item item-input">
            <input ng-model="form.lastName"
                    type="text"
                    placeholder="Last name">
        </label>
        <label class="item item-input">
            <input ng-model="form.email"
                    type="email"
                    placeholder="Email">
        </label>
        <label class="item item-input">
            <input ng-model="form.password"
                    type="password"
                    placeholder="Password">
        </label>
    </div>
    <div class="text-right padding-horizontal">
        <button class="button button-positive"
                ng-click="handleFormSubmit()">
            Submit
        </button>
    </div>
    <div class="text-center"
            ng-if="loading.submit">
        <ion-spinner></ion-spinner>
    </div>
    <p ng-if="submitted && !loading.submit"
        class="text-center">
        Submitted!
    </p>
</ion-content>
```

Ohjelma 5. *Ionic-kehyksellä toteutetun testiohjelman ensimmäisen näkymän HTML-kuvaus.*

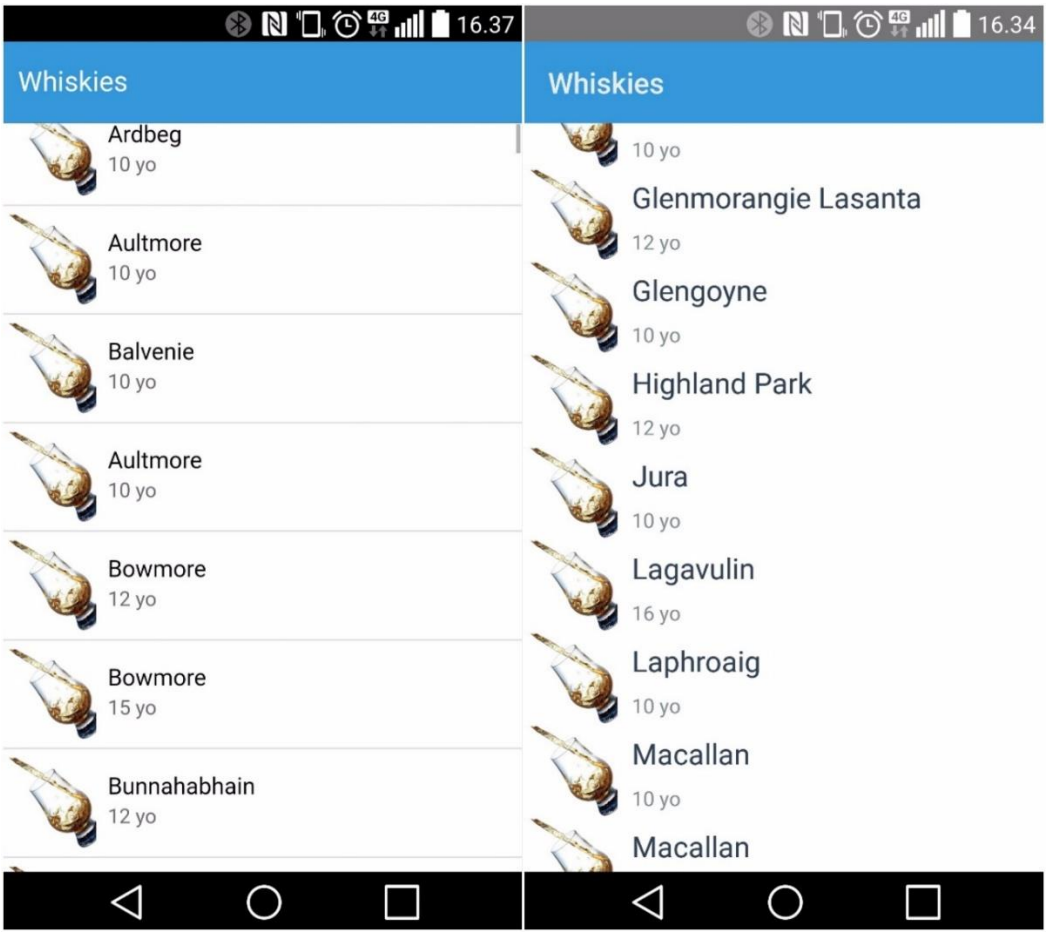
React Nativen osalta näkymä rakennettiin kehyksen tarjoamilla natiiveilla komponenteilla. Kyseinen näkymä hyödyntää pääasiassa TextInput -nimistä komponenttia, joka luo kentän, johon on mahdollista syöttää tekstiä. Ohjelmassa 6 nähdään lyhennetty toteutus React Native -testisovelluksen lomakkeesta. Ionic-toteutuksesta poiketen lomakkeen tekstikentän arvon päivittyessä joudutaan muuttunut arvo päivittämään onChangeText-tapahtuman avulla luokan tilaan.

```
<View>
      <View>
            <TextInput
                  ref={component => this._firstNameInput = component}
                  autoCapitalize="words"
                  name="firstName"
                  placeholder="First name"
                  onChangeText={(firstName) => this.setState({firstName})}
                  value={this.state.firstName}
                  editable={this.state.formEditable} />
      </View>
      <View>
            <TextInput
                  ref={component => this._lastNameInput = component}
                  autoCapitalize="words"
                  name="lastName"
                  placeholder="Last name"
                  onChangeText={(lastName) => this.setState({lastName})}
                  value={this.state.lastName}
                  editable={this.state.formEditable} />
      </View>
      <View style={styles.buttonContainer}>
            <Button
                  onPress={this.onFormSubmit.bind(this)}
                  buttonText="Submit">
            </Button>
      </View>
      <View style={styles.messageContainer}>
            {loadingIndicator}
            {submittedMessage}
      </View>
</View>
```

Ohjelma 6. *Lyhennetty versio React Native -testiohjelman ensimmäisen näkymän JSX-ohjelmakoodista.*

Testiohjelman toinen näkymä koostuu listasta, johon mallinnetaan yli 500 alkion JSON-tiedosto (Kuva 11). Pitkän listan avulla on mahdollista vertailla sovellusten tehokkuutta, tapauksessa, jossa ruudulle on tarve mallintaa suuri määrä elementtejä yhtäaikaisesti. Suorituskykyongelmat ilmenevät tällöin usein ruudunpäivitysnopeuden putoamisesta johtuvana animaation kankeutena ja ovat huomattavissa jopa paljaalla silmällä listaa vierittäessä. Kumpikin testattu ohjelmistokehys tarjoaa listaa varten komponentin, joka mallintaa automaattisesti ruudulla näkyvien elementtien lisäksi ainoastaan osan listasta, sen

sijaan, että koko lista mallinnettaisiin kerralla. Listaa vierittäessä vanhoja elementtejä tuhotaan ja uusia mallinnetaan automaattisesti, mikä parantaa suorituskykyä varsinkin, jos listassa olevien elementtien määrä on suuri.



Kuva 11. *Ionic- ja React Native -testisovellusten listanäkymä*

Ionic-kehyksen tapauksessa dynaaminen listakomponentti luodaan antamalla toistettavalle elementille *collection-repeat*-attribuutti. Ohjelmassa 7 nähdään näkymässä näkyvän listan toteutus HTML-merkkauskielellä. Vaihtoehtoinen tapa luoda lista on käyttää AngularJS:n tarjoamaa *ng-repeat*-attribuuttia, joka mallintaa listan kokonaisuudessaan, vaikka suurin osa listasta ei näkyisi ruudulla. Suorituskyvyn kannalta olennaista on testisovelluksen tapauksessa one-time binding-tekniikan käyttö. Pitkien listojen tapauksessa AngularJS:n tarkkailijoiden määrä kasvaa nopeasti suureksi, huonontaen suorituskykyä merkittävästi. Testisovelluksen tapauksessa two-way data binding- ja *ng-repeat*-toteutusta käyttäessä 500 alkion lista vaatisi 1001 (2 x 500 + 1) tarkkailijaa. Mikäli listassa olisi tarpeen näyttää JSON-alkiosta kahden parametrin sijaan esimerkiksi neljä, ylitettäisiin tällöin suosituksena oleva 2000 tarkkailijan raja. Optimoidun esimerkkiohjelman tapauksessa tarkkailijoita muodostuu vain n. 20-30 riippuen ruudulle mahtuvien elementtien määrästä. Tarkkailijoiden määrä saatiin siis oikeaoppisella toteutuksella pudotettua n. 2-3 %:iin alkuperäisestä määrästään.

```
<ion-content>
    <ion-refresher on-refresh="refreshList()">
    </ion-refresher>
    <div class="list">
        <a class="item item-avatar"
           collection-repeat="item in data"
            <img class="avatar-image"
                 src="http://dippa-cdn.s3.eu-central-1.amazo-
naws.com/whisky.png">
            <h2>{{::item.name}}</h2>
            <p>{{::item.age}} yo</p>
        </a>
    </div>
</ion-content>
```

Ohjelma 7. *Ionic-kehyksen dynaaminen lista.*

React Native tarjoaa dynaamisen listan luontiin niin ikään oman komponenttinsa nimeltään *ListView*, jonka käyttöä käsitellään ohjelmassa 8. Lista on tarvittaessa mahdollista luoda yhtenä komponenttina, mutta uudelleenkäytettävyyden vuoksi se on jaettu kahteen itsenäiseen osioonsa. Ylempi *PullToRefresh*-komponentti mahdollistaa listan päivittämisen vierittämällä yli listan yläosasta. Varsinainen listan toteuttava *ListView*-komponentti asetetaan tämän sisään ja määritetään *renderRow*-attribuutin avulla listan yksittäisen rivin mallintava komponentti.

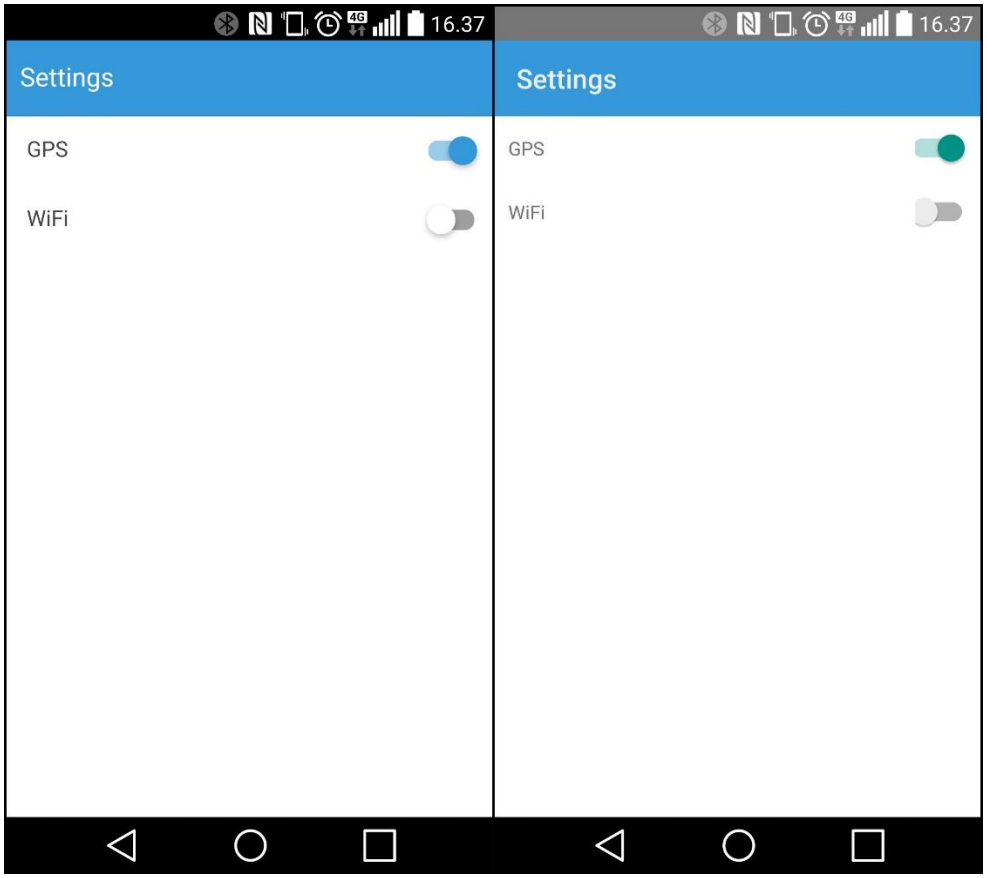
```
{/*
Parent view which calls the TouchableHighlight-component through renderRow.
*/}
<PullToRefreshViewAndroid
  style={styles.pullToRefresh}
  refreshing={this.state.isRefreshing}
  onRefresh={this._onRefresh.bind(this)}>
  <ListView
    dataSource={this.state.dataSource}
    renderRow={this._renderRow}
    style={styles.listView}>
  </ListView>
</PullToRefreshViewAndroid>

{/* The actual row rendering component called by ListView. */}
<TouchableHighlight onPress={_pressRow}
  underlayColor='#3498db'>
  <View style={styles.container}>
    <Image
      source={{uri:'http://dippa-cdn.s3.eu-central-1.amazo-
naws.com/whisky.png'}}
      style={styles.thumbnail}
    />
    <View style={styles.rightContainer}>
      <Text style={styles.name}>{item.name}</Text>
      <Text style={styles.age}>{item.age} yo</Text>
    </View>
  </View>
</TouchableHighlight>
```

Ohjelma 8. *Lyhennetty versio React Native -testisovelluksen listanäkymästä.*

React Native tarjoaa *ListView*-komponentin lisäksi Ionic-testisovelluksen *ng-repeat*-tekniikan kanssa vastaavasti koko listan samanaikaisesti mallintavan *ScrollView*-komponentin. React Nativen tapauksessa suorituskykyongelmat eivät aiheudu Ionic-testisovelluksen tavoin tarkkailijoista, vaan käyttöliittymän päivitystä viivästävästä sovelluslogiikasta. *ListView* tarjoaa tätä varten kaksi suorituskykyparannusta: ainoastaan muuttuneiden rivien uudelleenmallintamisen sekä mahdollisuuden säätää mallinnettavien rivien määrää yksittäisen tapahtumasilmukan kierroksen aikana. Koska tapahtumasilmukalla on n. 16,67ms aikaa valmistua, on mallinnettavien rivien määrää säätämällä mahdollisuus vaikuttaa merkittävästi suorituskykyyn, mikäli lista on pitkä.

Kuvassa 12 esitellään testisovellusten kolmas näkymä, joka ei toteutuksen puolesta esittele suorituskyvyn kannalta uusia toteutustapoja kahden aiemmin esitellyn näkymän lisäksi. Näkymä toteutettiin kuitenkin esittelemään kummankin ohjelmistokehyksen komponenttien toteutusta osana työssä toteutettua käyttäjätutkimusta.



Kuva 12. *Ionic- ja React Native -testisovellusten kolmas näkymä.*

Ohjelmassa 9 nähdään Ionic-testisovelluksen kolmannen näkymän mallintava HTML-kuvaus. Näkymässä hyödynnetään Ionic-ohjelmistokehyksen tarjoamaa *ion-toggle*-komponenttia, joka mallintaa näkymässä esiintyvät Androidin natiivia *Switch*-komponenttia muistuttavat kytkimet. Kytkimien päällä/pois -tilaa ohjataan jälleen AngularJS:n two-

way data binding -tekniikan avulla *status*-objektin gps- ja wifi -ominaisuuksien arvojen kautta.

```
<ion-content>
    <ion-toggle ng-model="status.gps"
                toggle-class="toggle-positive">
        GPS
    </ion-toggle>
    <ion-toggle ng-model="status.wifi"
                toggle-class="toggle-positive">
        WiFi
    </ion-toggle>
</ion-content>
```

Ohjelma 9. *Ionic-testisovelluksen kolmannen näkymän HTML-kuvaus*

Ohjelmassa 10 esitellään React Native -testisovelluksen vastaava JSX-kielinen toteutus. Pystysuuntaiseksi listaksi mallinnettavat komponentit on asetettu *ScrollView*-komponentin sisään, mikä mahdollistaa näkymän vierittämisen, mikäli kaikki mallinnetut komponentit eivät mahdu ruudulle kerralla. React Native tarjoaa vastaavasti *Switch*-komponentin kytkimen mallintamiseen. Kytkimien tilaa hallitaan komponentin *state*-objektin kautta, jonka päivitystä pyydetään aina *onValueChange*-attribuutin avulla kytkintä liikutettaessa.

```
<ScrollView>
  <View style={styles.row}>
    <Text style={styles.option}>GPS</Text>
    <View style={styles.rightContainer}>
      <Switch
        onValueChange={(value) => this.setState({gpsSwitchState: value})}
        style={styles.switch}
        value={this.state.gpsSwitchState} />
    </View>
  </View>
  <View style={styles.row}>
    <Text style={styles.option}>WiFi</Text>
    <View style={styles.rightContainer}>
      <Switch
        onValueChange={(value) => this.setState({wifiSwitchState: value})}
        value={this.state.wifiSwitchState} />
      </View>
  </View>
</ScrollView>
```

Ohjelma 10. *React Native -testisovelluksen kolmannen näkymän käyttöliittymäkomponenttien JSX-kuvaus.*

Testisovellusten toteutuksen kannalta vaikeuttava tekijä oli yhtenäisten komponenttien löytäminen, jotta sovellukset olisivat toistensa kanssa vertailukelpoisia. React Native -ohjelmistokehyksen Android-tuki julkaistiin vasta 14.9.2015, joten osalle yleisesti käytössä olevista komponenteista, kuten *Modal* ja *Slider*, ei vielä löydy olemassa olevaa Android-toteutusta.

4.2 Kriteeristö

Ollakseen varteenotettava vaihtoehto natiivisovellukselle, hybridisovelluksen tulee pystyä mittauksissa saman suuruusluokan tuloksiin natiivisovelluksen kanssa. Riippuen kriteeristä suorituskyvyn tulee olla tätäkin parempi, sillä esimerkiksi latausajat 100ms ja 900ms ovat samaa suuruusluokkaa, mutta jälkimmäinen mittaustulos ilmaisee tässä kontekstissa selkeästi riittämätöntä suorituskykyä.

Koska tarkasteltavaa sovellusta käytetään mobiililaitteella, on yksi merkittävimmistä sovellukselle asetettavista kriteereistä resurssien käyttö. Mobiililaitteiden merkittävä rajoite on fyysinen koko ja paino, mikä rajoittaa laitteeseen asennettavien komponenttien, kuten akun ja suorittimen valintaa. Suuri resurssien käyttö pienentää akun kestoa ja vaatii esimerkiksi suorittimen jäähdytykseltä enemmän. Tämän lisäksi sovelluksen käyttäjät saattavat lopettaa sovelluksen käytön, mikäli tämän huomataan käyttävän virtaa tavanomaista enemmän. Akun käytön lisäksi tarkasteltavat resurssit ovat sovelluksen vaatima tallennustila, keskusmuistin määrä sekä suoritinkuorma.

Toinen testisovellukselle asetettava kriteeri on latausaika, joka kuluu sovelluksen näkymää vaihtaessa sekä sovellusta avatessa sekä taustalta, että kokonaan suljetusta tilasta. Natiivisovellukset ovat tällä osa-alueella yleisesti tehokkaampia, sillä alustalle käännetyn ohjelmakoodin suorittaminen on luonnostaan tehokkaampaa kuin ajonaikaisesti tulkitun komentosarjakielellä kirjoitetun ohjelman. Nykypäivänä käyttäjät usein olettavat latausaikojen olevan lähes olemattomia, tai ne koetaan häiritseviksi käyttäjäkokemuksen kannalta. Tutkimusten mukaan 40 % käyttäjistä keskeyttävät näkymän latauksen ja siirtyvät taaksepäin, mikäli näkymä ei lataudu kolmen sekunnin aikana [36]. Testisovelluksen pitäisi tämän vuoksi kyetä alle kolmen sekunnin avautumisaikoihin.

Kolmas tarkasteltava kriteeri on käyttäjän subjektiivinen näkemys hybridisovelluksen toiminnasta. Koska JavaScript-ohjelmakoodia tulkitaan ajonaikaisesti, saattaa sovelluksen suorituskyky vaihdella tilanteen mukaan. Animaatiot eivät välttämättä ole sulavia ja sovellus ei aina reagoi käyttäjän syötteisiin viiveettä.

Yllä mainittujen kriteerien lisäksi merkittäviä kriteerejä ovat myös alustariippumattoman ohjelmistokehyksen aiheuttamat mahdolliset rajoitteet käyttöliittymäsuunnittelussa, toteutusvaiheen haastavuus sekä toteutuksen hinta [16]. Kyseiset kriteerit on jätetty tämän työn ulkopuolelle, mutta ne tulisi huomioida valittaessa sovelluksen toteutukseen käytettäviä tekniikoita.

4.3 Mittaustavat

Testisovellusten suorituskykyä mitattiin sekä kvantitatiivisin että kvalitatiivisin mittauksin. Kahta eri mittauslajia yhdistämällä selvitettiin, ovatko suorituskykyerot havaittavissa vain tarkoilla mittauksilla, vai ovatko suorituskykyerot havaittavissa myös käytännössä

sovellusta käyttäessä. Jokaisessa testissä käytettiin testilaitteena LG D855 -puhelinta Android-versiolla 5.0.0. Ionic-testisovelluksen mittauksissa käytetty Android-käyttöjärjestelmän WebView:n versio oli 48.0.2564.106.

Testisovelluksen vaatimaa tallennustilaa mitattiin tarkastelemalla sekä sovelluksen asennuspaketin, että asennetun sovelluksen kokoa puhelimen muistissa. Tulokset kirjattiin ylös 0,01 Mt tarkkuudella.

Testisovelluksen aiheuttamaa suoritinkuormaa mitattiin Android Debug Bridge -työkalun [30] avulla käyttämällä vmstat-komentoa. Mittaus suoritettiin kummallekin testisovellukselle kymmenen kertaa, yhden mittauksen kestäessä kolme minuuttia. Mittaus aloitettiin käynnistämällä vmstat-työkalun mittaus ja käynnistämällä 10 sekunnin jälkeen testisovellus testilaitteen työpöydällä sijaitsevasta pikakuvakkeesta. Mittauksen aikana testisovellukselle ei annettu muita syötteitä, sovelluksen avaamisen lisäksi. Mittausten aikana muita sovelluksia ei ollut avoinna. Testilaitteen mobiilidata-, Bluetooth-, GPS- ja NFC-yhteydet ja synkronointi-toiminto pidettiin poiskytkettyinä kaikkien mittausten ajan. WLAN-yhteys pidettiin käynnissä mittausten ajan, sillä testisovellusten listanäkymän testidata noudetaan verkon ylitse sovelluksen käynnistyessä. Mittausten on myös tarkoitus edustaa mahdollisimman paljon todellista käyttöä, jolloin verkkoyhteys on usein avoinna. WLAN-yhteyttä päätettiin käyttää mittauksissa mobiilidatan sijaan, jotta verkkoyhteyden laatuvaihteluista aiheutuvat vaikutukset mittaustuloksiin saataisiin minimoitua.

Sovelluksen aukeamiseen työpöydältä päänäkymään kuluvaa aikaa mitattiin kaappaamalla videokuvaa testilaitteen ruudusta Android Debug Bridge -työkalun screenrecord-komennolla. Testi suoritettiin kummallekin testisovellukselle kymmenen kertaa ja tulokseksi kirjattiin mittausten keskiarvo. Latautumisajaksi merkittiin aika, joka kului sovelluksen kuvakkeen painamisesta sovelluksen päänäkymän kaikkien elementtien latautumiseen. Sovelluksen kuvakkeen painallushetki taltioitiin hyödyntämällä testilaitteen kehitystyökalujen ”näytä painallukset” -toimintoa.

Sovelluksen vaatimaa keskusmuistin määrää mitattiin System Monitor -nimisellä Android sovelluksella [32]. Mittaus aloitettiin testilaitteen työpöydältä ja ennen testisovelluksen käynnistämistä kirjattiin ylös käytössä olevan muistin määrä. Testisovellus käynnistettiin pikakuvakkeesta ja tätä käytettiin yhdessä mittausjaksossa kolmen minuutin ajan. Mittausjaksolta kirjattiin tulokseksi korkein muistinkäyttöaste. Mittaus suoritettiin kymmenen kertaa kummallekin testisovellukselle ja tuloksista laskettiin keskiarvo.

Kvantitatiivisista mittauksista saadut tulokset ovat tärkeitä sovelluksen teknisen laadun kannalta. Sovelluksen käyttäjän kannalta tärkeä mittari on kuitenkin sovelluksen suorituskyvyn tason tarjoama käyttäjäkokemus. Testisovellusten laatua käyttäjän näkökulmasta mitattiin käyttäjätesteillä, joissa käyttäjää pyydettiin käyttämään kumpaakin testisovellusta ja vastaamaan asetettuihin kysymyksiin. Käyttäjätestien tarkoituksena on sel-

vittää ovatko testisovellusten suorituskykyerot havaittavissa myös paljaalla silmällä tavallisen käyttäjän näkökulmasta. Käyttäjätesteissä esitetyt kysymykset on annettu liitteessä A.

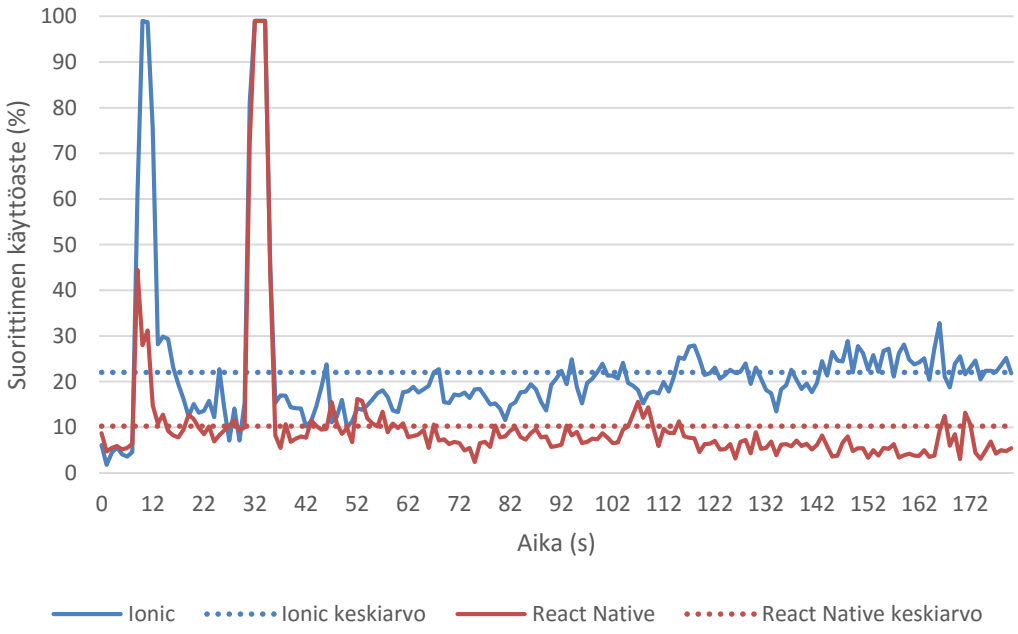
5. TULOKSET JA ARVIOINTI

Tässä luvussa esitellään ja analysoidaan testisovelluksille suoritettujen testien tulokset mittausmenetelmittäin. Luvun lopuksi arvioidaan suoritettujen mittausten luotettavuutta ja esitellään tulosten pohjalta saadut johtopäätökset. Luvussa käydään läpi myös työn jälkeen heränneet kehitysehdotukset.

5.1 Suorittimen käyttöaste

Kuvassa 13 käsitellään kummankin testisovelluksen aiheuttamaa suoritinkuormaa vmstat-työkalun mittauksen aikana. Kuvaajat on piirretty kummankin testin mittaustulosten (Liite B, Liite D) keskiarvosta kullakin testin sekunnilla. Vmstat-työkalun mittaama käyttöaste pyöristyy lähimpään prosenttiyksikköön, joten mittausvirhe on suoritinmittauksissa 0,5 prosenttiyksikköä. Mittausjaksokohtaiset kuvaajat on annettu liitteissä C ja E.

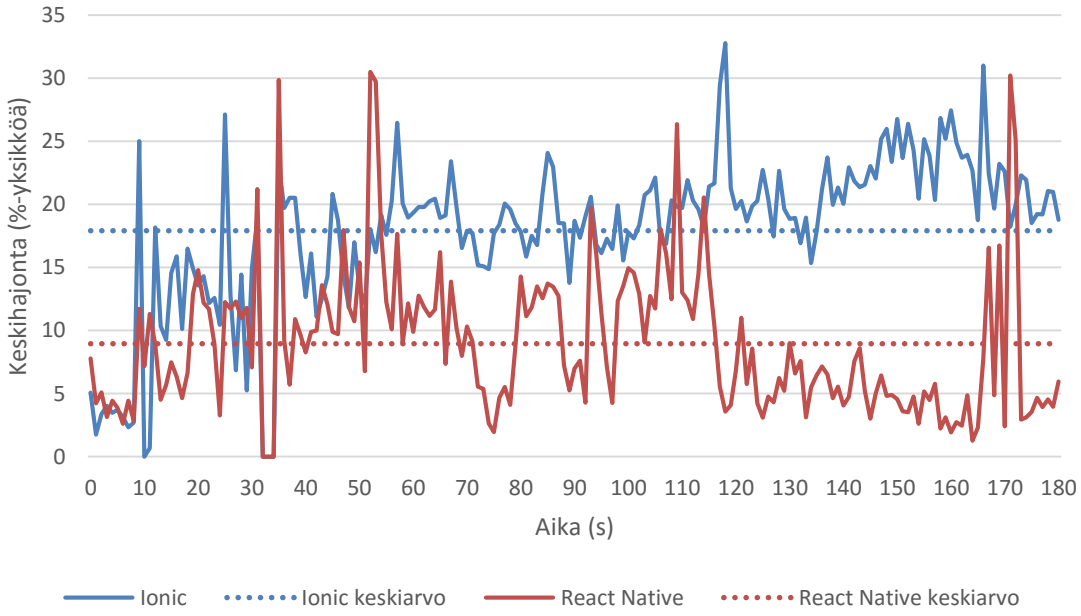
Kuvaajista nähdään selkeästi sovelluksen käynnistysajankohta kohdassa 10s. Ionic-testisovelluksen tapauksessa suorittimen käyttöaste nousee selkeästi lähes 100 % tasolle, React Native -testisovelluksen jäädessä keskimäärin hieman alle 45 % käyttöasteen. Mittausjakson aikavälillä 32-37s nähdään toinen piikki kummankin sovelluksen suoritinkäytössä. Testeissä huomattiin piikin ajoittuvan samaan ajanhetkeen, jolloin testilaitteen näytön taustavalo himmeni pienimmälle teholleen. Taustavalon himmenemistä ei ollut mahdollista testilaitteessa estää ja tapahtuma toistui jokaisessa testissä samalla ajanhetkellä, joten testituloksia ei tästä johtuen ollut tarvetta hylätä.



Kuva 13. *Testisovellusten suoritinkäyttö ajan suhteen sovellus etualalla*

Ionic-testisovelluksen suoritinkäytön keskiarvo on koko mittausjakson ajalta 22,02 % React Nativen jäädessä 10,22 % tasolle. Sovellusten suoritinkäytössä on siis huomattavissa selkeä ero React Nativen eduksi.

Kuvassa 14 tarkastellaan testisovellusten suoritinkäytön keskihajontaa mittausjaksojen välillä. Kuvaajasta nähdään, että Ionic-testisovelluksen suoritinkäyttö vaihtelee selkeästi eniten eri mittausjaksoilla. React Native -testisovelluksen suoritinkäyttö on selkeästi tasaisempaa, mikä nähdään myös liitteen E kuvaajista. Tarkasteltaessa keskihajonnan keskiarvoa kaikkien mittausjaksojen ajalta, nähdään Ionic-testisovelluksen asettuvan 17,90 prosenttiyksikön kohdalle, React Nativen keskihajonnan keskiarvon jäädessä 8,96 prosenttiyksikön tasolle.

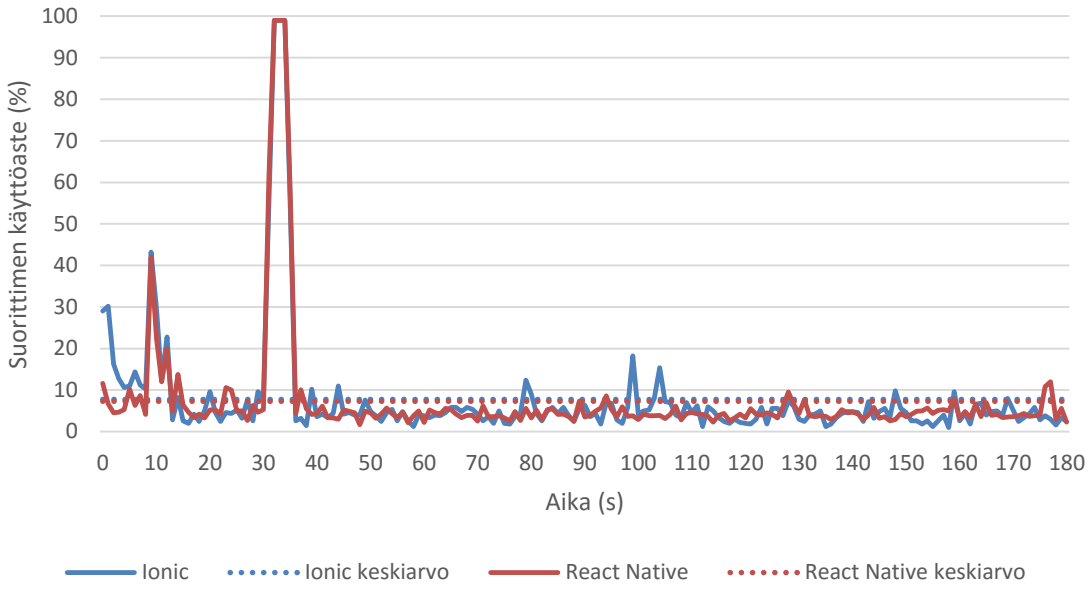


Kuva 14. *Testisovellusten suoritinkäyttöasteen keskihajonta sovellus etualalla*

Tarkasteltaessa kuvaajia yhdessä nähdään React Native -testisovelluksen olevan suoritinkäytön näkökulmasta jonkin verran kevyempi toteutus. Koska kummankin testisovelluksen taustalogiikka on kirjoitettu JavaScript-ohjelmointikielellä, johtuvat erot todennäköisesti valtaosin käyttöliittymäkomponenttien mallinnuksesta. Sovelluksen käynnistyessä ilmenevien suoritinkäytön piikkien erot tukevat myös tätä päätelmää, sillä kyseisellä hetkellä käyttöliittymä mallinnetaan ruudulle kokonaisuudessaan. Testien pohjalta React Nativen natiivit käyttöliittymäkomponentit ovat selkeästi kevyempiä mallintaa, verrattuna Ionic-testisovelluksen HTML5-elementteihin. Ionic-testisovelluksen tulokset saattavat kuitenkin vaihdella riippuen käytettävästä testilaitteesta. WebView-näkymä pohjautuu oletuksena laitealustan tarjoamaan natiiviselaimeen, minkä vuoksi tulokset saattavat testilaitteesta riippuen huonontua tai parantua selkeästi Ionic-testisovelluksen osalta.

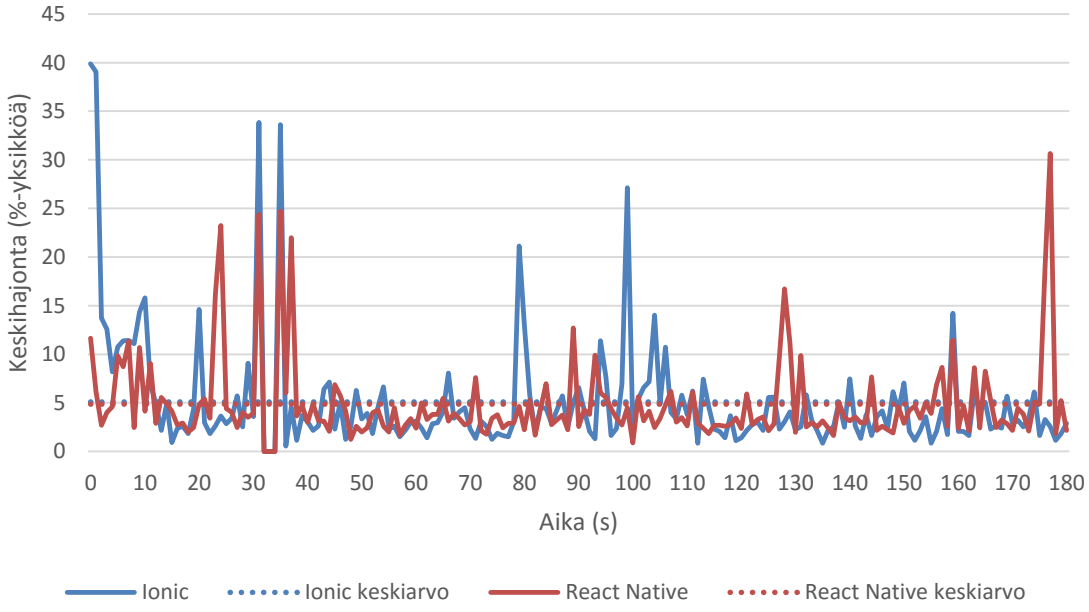
Kuvassa 15 vertaillaan testisovellusten suoritinkäyttöä sovelluksen ollessa auki taustalla. Kuvaajista näemme, että kummankin testisovelluksen kohdalla suoritinkäyttö on tasaista

verrattuna aiempaan testiin sovellus etualalla. Tarkasteltaessa suorittimen käyttöasteen keskiarvoa kaikilta mittausjaksoilta nähdään myös, että käyttöasteet ovat lähellä toisiaan. Ionic-testisovelluksen keskimääräinen suorittimen käyttöaste oli 7,72 % ja React Native -testisovelluksen 7,29 %.



Kuva 15. *Testisovellusten suoritinkäyttö sovelluksen ollessa auki taustalla*

Tarkasteltaessa testisovellusten suoritinkäytön keskihajontaa (Kuva 16) nähdään, että hajonnan keskiarvo on lähes samalla tasolla kummallakin testisovelluksella. Ionic-testisovelluksen keskihajonnan keskiarvo kymmenellä mittausjaksolla oli 5,08 prosenttiyksikköä ja React Native -testisovelluksen 4,85 prosenttiyksikköä.

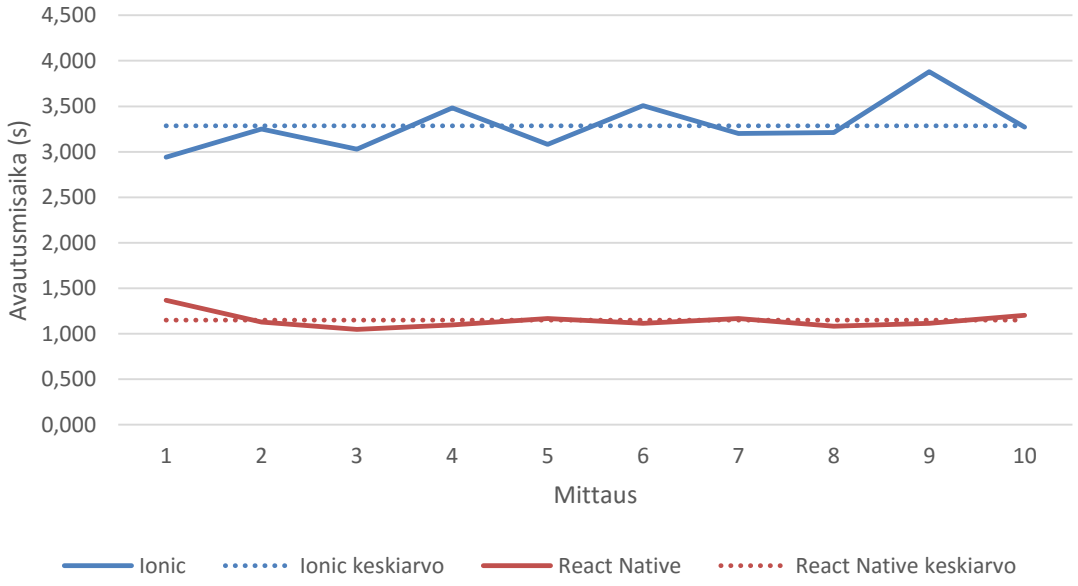


Kuva 16. *Testisovellusten suoritinkäytön keskihajonta sovelluksen ollessa auki taustalla*

Sovelluksen tausta-ajotestien perusteella voidaan todeta, että suoritettaessa sovellusta taustalla toteutustekniikan valinnalla ei ole merkittävää vaikutusta. Tuloksissa nähdään pieni ero React Native -testisovelluksen eduksi, mutta tämä jää vielä mittausvirheen sallimiin rajoihin. Mikäli testattavissa sovelluksissa olisi erillistä sovelluksen taustalla ollessa suoritettavaa toiminnallisuutta, olisi teknologioiden välillä mahdollisesti ollut suurempia eroja. Myös testilaitteen käyttöjärjestelmän virransäästöominaisuudet vaikuttavat kyseisen testin tulokseen. Testisovellusten välillä saattaa olla eroja käytettäessä vanhempaa Android-käyttöjärjestelmäversiota, jossa virransäästöominaisuudet eivät ole yhtä kehittyneet huomioimaan taustalla avoinna olevia sovelluksia.

5.2 Sovelluksen avautumisaika

Kuvassa 17 tarkastellaan sovellusten avautumisaikaa testilaitteen työpöydältä suljetusta tilasta sovelluksen päänäkymään. Mittausten pohjana käytetyn ruudunkaappausvideon ruudunpäivitysnopeus oli 30 ruutua sekunnissa. Tämän pohjalta laskettuna tulosten mittausvirhe on noin 3,3 %.



Kuva 17. *Testisovellusten avautumisaika työpöydältä*

Kuvaajista nähdään React Nativen suoriutuvan testistä selkeästi paremmin. Ionic-testisovelluksen avautumisajan keskiarvo on kymmenen mittausjakson pohjalta 3,285 s, React Nativen ollessa keskimäärin 2,136 sekuntia nopeampi keskiarvolla 1,149 s. Mittausten tulokset ovat tarkasteltavissa liitteessä F.

Jos sovelluksen avautumisaika on merkittävä kriteeri toteutettavassa sovelluksessa, on valinta toteutusteknologioiden välillä mittausten perusteella selkeä. Ionic-testisovelluksen avautumisaika ylittää kriteeriksi asetetun kolmen sekunnin rajan yhtä mittausjaksoa lukuun ottamatta. Usein toistuvassa käytössä Ionic-sovelluksen avautumisajan pituus

koetaan todennäköisesti häiritseväksi. Avautumisajan koettua pituutta on tilanteesta riippuen kuitenkin mahdollista kompensoida, esimerkiksi näyttämällä käyttäjälle latauksen aikana esimerkiksi osa latautuvasta käyttöliittymästä.

5.3 Keskusmuistin käyttö

Kymmenen mittausjakson aikana Ionic-testisovellus nosti keskusmuistin käyttöastetta 1,36 prosenttiyksikköä verrattuna keskusmuistin käyttöasteeseen testilaitteen työpöydällä. React Native -testisovellus puolestaan nosti käyttöastetta 2,68 prosenttiyksikköä, eli 1,32 prosenttiyksikköä enemmän kuin Ionic-testisovellus. Mittausjaksojen tulokset on annettu liitteessä G. Mittaustulokset kirjattiin 1 Mt tarkkuudella, joten mittausvirhe oli ±0,5 Mt eli ±0,02 prosenttiyksikköä.

Mittausten aikana huomattiin, että React Nativen muistinkäyttö nousi selkeästi vierittäessä listanäkymää. Listanäkymää vierittäessä ensimmäistä kertaa listan loppuun muistinkäyttö nousi noin 60 Mt eli 2,10 prosenttiyksikköä. Valtaosa React Native -testisovelluksen muistinkäytöstä siis muodostui vasta listaa vierittäessä. Mittausten aikana huomattiin myös, että listan päivittäminen nosti muistinkäyttöä hetkellisesti. Ionic-testisovelluksella listan vierittäminen tai päivittäminen ei vaikuttanut huomattavasti muistinkäytön tasoon.

Kummankaan testisovelluksen muistinkäyttö ei suoritettujen mittausten perusteella riitä tärkeäksi valintakriteeriksi toteutusteknologiaa valitessa. Sovelluksen vaatiman keskusmuistin määrä nousee tärkeäksi kriteeriksi vasta, kun keskusmuistin on mahdollista loppua sovellusta käyttäessä. Muistin loppuessa joudutaan keskusmuistin lisäksi osa sovellusten vaatimasta muistista kirjoittamaan laitteen massamuistiin. Massamuistin luku- ja kirjoitusoperaatiot ovat keskusmuistia hitaampia, mikä useimmiten aiheuttaa hitautta myös sovelluksen tai koko laitteen käyttöjärjestelmän toimintaan.

5.4 Tallennustila

Toteutetun Ionic-testisovelluksen asennuspaketin koko oli 2,83 Mt. React Native -testisovelluksen asennuspaketti oli selkeästi suurempi ja vaati 7,37 Mt tallennustilaa. Testilaitteeseen asennettuna Ionic-testisovellus vaati 3,15 Mt tallennustilaa ja React Native -testisovellus 20,54 Mt. Kumpikaan testisovellus ei sisältänyt käyttöliittymässä esitettäviä kuvatiedostoja, vaan nämä ladattiin verkkoyhteyden yli sovelluksen käynnistyessä. Sovelluksen koko koostui siis ainoastaan ohjelmistokehysten omista tiedostoista, sekä toteutuksen aikana luoduista ohjelmakooditiedostoista.

Tulosten pohjalta nähdään React Native -ohjelmistokehyksen vaativan selkeästi enemmän tallennustilaa jo yksinkertaisen sovelluksen toteutuksella, mikä johtuu todennäköisesti itse ohjelmistokehyksen suuremmasta koosta. Ionic-testisovellus puolestaan vaati vain n. 15 % React Native -testisovelluksen tarvitsemasta tallennustilasta, minkä vuoksi Ionic-ohjelmistokehyksen valinta kannattaa, mikäli tallennustila on merkittävä kriteeri

toteutettavalla sovellukselle. Huomioitavaa kuitenkin on, että erot toteutusten välillä todennäköisesti kapenevat sovelluksen ohjelmakoodin määrän kasvaessa, jolloin ohjelmistokehyksen vaatiman tallennustilan osuus pienenee suhteessa muuhun sovellukseen.

5.5 Subjektiiviset tulokset

Suurin osa käyttäjätesteihin osallistuneista henkilöistä koki Ionic-testisovelluksen animaatioiden kankeuden häiritsevän jonkin verran sovelluksen käyttöä. Yksikään testihenkilöistä ei kuitenkaan nähnyt tämän estävän sovelluksen normaalia käyttöä, vaan ilmoittivat kykenevänsä käyttämään sovellusta lähes ongelmitta. React Native -testisovelluksen animaatiot koettiin sulaviksi ja sovelluksen vastaavan hyvin käyttäjän syötteisiin. Suurin osa testihenkilöistä mainitsi Ionic-testisovelluksen animaatioiden heikomman suorituskyvyn vasta saatuaan vertailtavaksi React Native -testisovelluksen. Tämän pohjalta voidaan todeta, että Ionic-testisovelluksen käyttäjäkokemus on kontekstista riippuvaa. Mikäli tarjolla on ominaisuuksiltaan vastaava sovellus, johon Ionic-sovellusta on mahdollista verrata, ei Ionic-ohjelmistokehyksellä toteutetun sovelluksen käyttöliittymän suorituskyky välttämättä ole käyttäjän näkökulmasta riittävä.

Kysyttäessä mitä käyttäjät parantaisivat testisovelluksen toiminnassa, esille nousi jonkin verran suorituskykyyn liittymättömiä asioita. Näkymien välillä pyyhkäisemällä liikkuminen ei ollut intuitiivista ja osa testikäyttäjistä ehdotti, että pyyhkäisyn mahdollisuutta kuvaamaan lisättäisiin nuolet. Yksi käyttäjistä oli sitä mieltä, että sivuttaissuuntainen pyyhkäisy näkymien välillä ei tunnu luonnolliselta, vaan navigointi tulisi olla mahdollista valikon tai välilehtien kautta. Osa käyttäjistä mainitsi, että Ionic-testisovelluksen animaatioiden sulavuutta voisi parantaa. React Native -testisovelluksella vastaavaa kehitysehdotusta ei maininnut kukaan testikäyttäjistä.

5.6 Testien ulkopuolisia huomioita

Ionic-testisovellusta testattiin kehitysvaiheessa myös Crosswalk-WebView-näkymää hyödyntäen. Käytössä olleella testilaitteella sovelluksen suorituskyvyn havaittiin heikentyvän verrattuna laitteessa oletuksena toimivan WebView-näkymän suorituskykyyn. Tämän vuoksi Ionic-testisovelluksen varsinaisessa testiversiossa käytettiin alustan oletusselaimeen pohjautuvaa WebView-näkymää. Useimmissa tapauksissa Crosswalkin on kuitenkin todettu parantavan sovelluksen suorituskykyä huomattavasti, varsinkin vanhempaa kuin Android versiota 4.0.3 käyttävillä laitteilla [35]. Crosswalkin ja muiden kolmannen osapuolen WebView-näkymien suorituskyky ei ollut tämän tutkimuksen keskipisteenä, joten asiaa ei tässä työssä käsitellä tämän syvällisemmin. Toteuttaessa Ionic-sovellusta, on kuitenkin aiheellista tutkia muiden WebView-näkymien suorituskykyä verrattuna laitteessa oletuksena toimivaan WebView-näkymään erityisesti, jos sovellusten potentiaalisten käyttäjien laitteiden käyttöjärjestelmäversioita ei tiedetä.

Testeissä ei varsinaisesti mitattu alustojen välillä uudelleenkäytettävän ohjelmakoodin määrää, sillä kaikki testit suoritettiin ainoastaan Android-alustalla. Alustariippumattoman teknologian kannattavuutta tarkastellessa on aina aiheellista arvioida, kuinka alustariippumaton toteutus todellisuudessa on kyseessä. Testisovelluksista React Native -sovellus vaatii pääohjelmansa osalta oman toteutuksen iOS-alustalle. Ionic-testisovelluksen osalta toteutus on teoriassa täysin alustariippumaton, mutta tämän työn puitteissa todellista yhteensopivuutta muilla alustoilla ei testattu.

Ionic-ohjelmistokehyksen iOS-yhteensopivuuden kanssa havaittiin puutteita tämän tutkimuksen ulkopuolisissa projekteissa. Joissain tilanteissa toteutetun sovelluksen Android-käännös on toiminut ongelmitta, mutta käännettäessä sama sovellus iOS-alustalle on suorittimen käyttöaste keskeytyksettä 100 % tasolla ja sovellus vuotaa muistia. Osan Ionic-ohjelmistokehykselle tai tämän lisäosille annettavista konfiguraatioista on myös havaittu olevan keskeneräisiä. Esimerkiksi sovelluksen latautuessa esitettävän käynnistysikkunan kuvan esitys- ja häivytysaikoihin vaikuttavat attribuutit eivät sovelluksen käännöksen jälkeen toimi, tai toimivat tavallisesta poikkeavasti. Joissain tapauksissa sovellus jättää käynnistysikkunan kuvan lataamatta, mikäli tälle asetettu esitysaika on liian pieni tai suuri riippumatta siitä, onko kuva asetettu näytettäväksi vai piilotetuksi. Edellä kuvatun tapaisia ongelmia on mahdollista esiintyä valitusta ohjelmistokehyksestä riippumatta, mutta ongelmien olemassaolo on syytä huomioida toteutusteknologiaa valittaessa. Ionic-ohjelmistokehyksen tapauksessa mainitut ongelmat johtivat sovelluksen kehityksen loppuvaiheessa käytetyn ohjelmistokehyksen vaihtoon, sillä ongelmat estivät sovelluksen toiminnan iOS-alustalla kokonaisuudessaan. React Native -ohjelmistokehyksen osalta vastaavia ongelmia ei toistaiseksi ole havaittu.

5.7 Tulosten luotettavuus

Määrällisten testien osalta saatuja tuloksia saattavat pääasiassa vääristää testilaitteen taustaprosessit mittausten aikana. Taustaprosessien vaikutusta pyrittiin minimoimaan suorittamalla mittaukset vuorotellen ja useampia kertoja. Mittaustulokset saattavat muuttua, mikäli mittaukset suoritetaan eri laitteella, jonka laitteisto, käyttöjärjestelmäversio tai asennetut sovellukset poikkeavat tässä työssä käytetystä testilaitteesta. Valtaosassa testejä saadut mittaustulokset testisovellusten välillä poikkesivat toisistaan kuitenkin niin merkittävästi, että näiden perusteella on mahdollista tehdä vertailua testattujen ohjelmistokehysten välillä.

Sovelluksen suorituskykyyn vaikuttaa luonnollisesti sovelluksen toteutuksen optimoinnin taso. Kummankin testisovelluksen toteutus pyrittiin toteuttamaan mahdollisimman tehokkaasti huomioiden luvussa 3 käsitellyt käytännöt. Testien mittaustulokset saattavat muuttua, mikäli vastaavanlaiset testisovellukset toteutetaan uudelleen eri henkilön tai henkilöiden toimesta.

Käyttäjätestien tulosten luotettavuuteen vaikuttaa ensisijaisesti testeihin osallistuneiden henkilöiden määrä sekä henkilökohtaiset ominaisuudet, kuten ikä, sukupuoli, kokemus mobiilisovellusten käytöstä tai kehityksestä sekä muut henkilökohtaiset mieltymykset koskien mobiilisovellusten käyttöä. Tässä työssä käyttäjätesteihin osallistuneiden henkilöiden pieni määrä saattaa vääristää saatuja tuloksia, mutta koska jokaisesta käyttäjätestistä saadut tulokset ovat hyvin yhteneviä, on näiden perusteella mahdollista tehdä suuntaa-antavia johtopäätöksiä. Käyttäjätestien tulokset tukevat myös määrällisistä mittauksista saatuja tuloksia, joten myös tämän perusteella voidaan käyttäjätestien tulosten päätellä olevan luotettavia.

5.8 Johtopäätökset

Tarkasteltaessa kvantitatiivisten testien tuloksia yhdessä nähdään, että React Native -testisovellus suoriutuu selkeästi paremmin ja saatavilla olevien resurssien käyttö on selkeästi tasaisempaa verrattuna Ionic-toteutukseen. React Native on myös käyttäjätestien voittaja, mutta ero tuloksissa ei ole yhtä helposti tulkittava, sillä Ionic-testisovelluksen puutteet ilmenivät lähes kaikissa testeissä vasta, kun sovellusta verrattiin suoraan vastaavaan React Native -toteutukseen.

Testien perusteella alustariippumattomat ohjelmistokehykset ovat useimmiten varteenotettava vaihtoehto natiiville toteutustavalle. Kumpikin testatuista ohjelmistokehyksistä on kykenevä riittävään suorituskykyyn kummallekin toteutustavalle ominaisten rajoitteiden puitteissa. Suurimmat rajoitteet hybriditoteutustavalle ovat käyttöliittymän edustavuus ja resurssien käyttö. Toteutustavan etuna ovat kuitenkin useimmiten nopea ja edullinen kehitys ja saatavilla olevien kehittäjien määrä, sillä useimmiten JavaScript-ohjelmointikieltä käyttäneet web-kehittäjät kykenevät kehittämään hybridisovelluksen ilman pitkäkestoista opiskelua. Hybridinatiivin toteutustavan suurimpana rajoitteena on yhteensopivien alustojen vähäinen määrä. Toteutustavan etuina ovat kuitenkin edulliset kehityskustannukset verrattuna natiiviin ja käyttöliittymän edustavuus verrattuna hybridiin toteutustapaan.

Taulukossa 4 käsitellään eri toteutustapojen soveltuvuutta suhteessa sovellukselle asetettuihin vaatimuksiin ja saatavilla oleviin resursseihin, kun sovellus on tarkoitus kehittää useammalle alustalle. Soveltuvuutta arvioidaan asteikolla 1-5, jossa numero 1 on huonoiten soveltuva ja 5 on parhaiten soveltuva toteutustapa asetetulla kriteerillä. Soveltuvuuden arvo esitetään taulukossa tähtinä. Tilanteessa, jossa toteutustavan käyttö ei ole lainkaan mahdollinen esitetään solussa viiva.

Kyseinen taulukko edustaa vain yhtä näkökulmaa kyseisten toteutustapojen soveltuvuudesta kuvailtuihin tilanteisiin. Tämän vuoksi taulukkoa tulee käyttää vain suuntaa-antavana ohjeena toteutustapaa valitessa. Arvioitaessa soveltuvinta toteutustapaa on myös yhdistettävä useampia vaatimuksia keskenään ja arvioitava toteutustavan soveltuvuutta uudelleen näiden yhdistelmänä.

Taulukko 4. *Toteutusteknologioiden soveltuvuus tilanteisiin, joissa sovellus toteutetaan useammalle alustalle*

Vaatimus	Natiivi	Hybridinatiivi	Hybridi	Web-sovellus
Tuki harvinaisille alustoille	***	-	**	*****
Graafisesti vaativa käyttöliittymä	*****	****	*	*
Käyttöliittymän edustavuus tärkeää	*****	*****	**	*
Työpöytätuki mobiilin lisäksi	*	**	***	*****
Laitteistotukea vaativia ominaisuuksia	*****	****	****	**
Kehitysaika	*	****	*****	*****
Kustannustehokkuus	*	****	****	*****
Laitteistoresurssit	Natiivi	Hybridinatiivi	Hybridi	Web-sovellus
Prosessori	*****	*****	**	**
Akku	*****	*****	**	*
Keskusmuisti	*****	*****	**	**
Tallennustila	**	**	****	*****
Henkilöstö	Natiivi	Hybridinatiivi	Hybridi	Web-sovellus
Olemassa olevia web-osaajia	*	****	*****	*****
Olemassa olevia natiivi-osaajia	*****	**	*	*
Vähän toteuttavia henkilöitä	*	*****	*****	*****
Nykyiset sovellukset	Natiivi	Hybridinatiivi	Hybridi	Web-sovellus
Olemassa oleva web-sovellus	*	***	******	***
Olemassa oleva natiivisovellus	*****	****	****	*
Olemassa oleva hybridinatiivi	*	*****	***	*
Olemassa oleva hybridi	*	***	*****	***

Mikäli toteutettava sovelluksen on tarkoitus tukea harvinaisempia alustoja, eli muita kuin Googlen Android- ja Apple iOS -käyttöjärjestelmiä, on usein kustannustehokkainta toteuttaa web-sovellus. Mikäli vaatimuksena on saada laitteeseen asennettava sovellus, on suositeltavaa toteuttaa lisäksi natiivi- tai hybridisovellus, jonka WebView-näkymässä suoritetaan ensin toteutettua web-sovellusta. Kyseisen toteutustavan etuna on, kustannustehokkuuden lisäksi, että web-sovellusta on samalla mahdollista käyttää myös työpöytäkäytössä tietokoneen selaimella. Vaihtoehtona on toteuttaa erillinen natiivisovellus jokaiselle vaaditulle alustalle. Kyseinen vaihtoehto sopii tilanteeseen, jossa sovelluksen ulkoasulta ja toiminnalta vaaditaan edustavuutta, mutta aika ja toteutuksen hinta eivät ole rajoittavia tekijöitä.

Jos toteutettava sovellus tarvitsee toiminnassaan runsaasti laitteiston tarjoamia ominaisuuksia, turvallisin toteutustapa on useimmiten natiivisovellus. Toteutukseltaan hybridit ja hybridinatiivit ohjelmistokehykset tukevat useimmiten valtaosaa laitteiston tarjoamista rajapinnoista, mutta joissain tapauksissa yhteensopivuusongelmia saattaa esiintyä todennäköisemmin verrattuna puhtaasti natiiviin toteutukseen. Osa alustariippumattomista ohjelmistokehyksistä tukee myös natiivisti toteutettuja ominaisuuksia alustariippumattoman osuuden rinnalla. Tämän vuoksi esimerkiksi React Nativen tapauksessa on mahdollista

kirjoittaa natiivi toteutus sille sovelluksen osalle, joka vaatii laitteistolta tukea ominaisuuteen, jota React Native ei suoraan tue.

Arvioitaessa toteutustapojen soveltuvuutta laitteistoresurssien perusteella natiivi ja hybridinatiivi vaihtoehto ovat useimmiten keskenään lähes tasaväkisiä ja parempia vaihtoehtoja verrattuna hybridi- tai web-sovellukseen. Tasaväkisyys johtuu siitä, että natiivin ja hybridinatiivin toteutukset ovat teknisesti hyvin lähellä toisiaan ja ero syntyy pääasiassa sovelluksen logiikan toteutuskielestä. Useimmiten hybridinatiivit ohjelmistokehykset vaativat myös jonkin verran natiivin ohjelmakoodin kirjoittamista, mikä kaventaa toteutustapojen eroja entisestään. Ainoa hybridi- ja web-sovelluksen eduksi katsottava resurssi on tallennustila, mutta tämän tarve on toteutettavan sovelluksen rakenteesta riippuvaa. Web-sovellus ei selaimen välimuistiin tallennettavia tilapäisiä tiedostoja lukuun ottamatta vaadi tallennustilaa lainkaan, mutta sovellus joudutaan lataamaan verkosta käytännössä jokaisella käyttökerralla uudelleen.

Saatavilla olevien toteuttavien henkilöiden kokemus kustakin teknologiasta vaikuttaa merkittävästi toteutustavan valintaan. Mikäli saatavuus web-teknologioiden osaajista on hyvä, suositeltavaa on usein toteuttaa ensisijaisesti hybridi- tai web-sovellus, sillä niiden osalta jo olemassa olevan JavaScript-, HTML- ja CSS-osaamisen hyödyntäminen on helpointa. Myös hybridinatiivia sovellusta toteutettaessa pystytään hyödyntämään JavaScript-osaamista vahvasti, mutta koska käyttöliittymäelementit ovat toteutukseltaan natiiveja jää HTML- ja CSS-osaaminen pääosin hyödyntämättä. Hybridinatiivin sovelluksen toteutuksessa joudutaan toisinaan myös kirjoittamaan osa sovelluksesta alustakohtaisesti natiivilla ohjelmointikielellä, mikä saattaa aiheuttaa jonkin verran lisäkustannuksia opiskeluna. Mikäli projektiin saatavilla olevaa henkilöstöä on vähän, on useimmiten kannattavaa toteuttaa sovellus mitä tahansa alustariippumatonta tekniikkaa käyttäen natiivin toteutuksen sijasta. Kannattavuus voidaan perustella sekä toteutukseen kuluvan ajan, että opiskelukustannusten perusteella. Mikäli toteuttavia henkilöitä on vähemmän, kuin alustoja, joille sovellus toteutetaan, ei sovelluksia ole käytännössä mahdollista toteuttaa natiivisti täysin rinnakkain. Alustariippumatonta toteutusta tehdessä voidaan useimmiten keskittyä rakentamaan sovellus ensin yhdelle alustalle ja tämän valmistuttua pienellä työllä varmistaa sovelluksen toimivuus myös muilla halutuilla alustoilla jo yhden henkilön projektina. Useimmiten on kuitenkin kannattavaa testata sovelluksen toimivuutta kaikilla toteutettavilla alustoilla pitkin projektia, sillä käytännössä sovellukset harvoin ovat täysin alustariippumattomia. Esimerkiksi React Native -ohjelmistokehys ei lupaa kirjoitushetkellä olevansa vielä täysin alustariippumaton, vaan vaatii myös jonkin verran alustakohtaista toteutusta.

Yksittäisen ohjelmistokehittäjän näkökulmasta alustariippumattomat ohjelmistokehykset ovat hyödyllisiä, sillä ne useimmiten poistavat tarpeen opiskella useampaa toteutuskieltä. Suosituimmat alustariippumattomat ohjelmistokehykset tukevat sovelluslogiikkansa osalta JavaScript-ohjelmointikieltä. Hybridisovellusten tapauksessa käyttöliittymän toteutukseen vaaditaan puolestaan HTML- ja CSS-osaamista, jotka JavaSciptin ohella ovat

useimmiten tuttuja erityisesti web-kehittäjille. Hybridinatiiveja sovelluksia kehittäessä JavaScript-kielen tai tämän syntaksilaajennuksen opettelu useimmiten riittää, sillä käyttöliittymä mallinnetaan kutsumalla alustan tarjoamien käyttöliittymäkomponenttien rajapintoja käytetyn ohjelmistokehyksen tarjoaman abstraktiokerroksen läpi.

Kehitysprojektin resursoinnin kannalta alustariippumattomat ohjelmistokehykset tarjoavat joustovaraa erityisesti, jos sovellusta kehittävä yritys on pieni tai tarjolla olevia kehittäjiä on muusta syystä vähän. Yhteen ohjelmakoodikantaan perustuva sovellus takaa sen, että projektissa työskentelevät kehittäjät voivat kehittää suoraan myös toistensa tekemiä ominaisuuksia, mikä ei ole mahdollista esimerkiksi kahteen eri ohjelmointikieleen perustuvien natiivisovellusten tapauksessa. Poikkeuksena tähän on tilanne, jossa projektissa useampi kehittäjä on useamman natiiviohjelmointikielen ja -työkalun asiantuntija. Tässäkin tapauksessa ohjelmakoodin parempi laatu on kuitenkin helpompi saavuttaa, mikäli käytettävissä olevat resurssit voidaan kohdistaa yhden ohjelmakoodikannan kehitykseen ja ylläpitoon.

Tämän työn tulosten perusteella ei ole enää kyse siitä soveltuvatko alustariippumattomat teknologiat mobiilikehitykseen natiivin ratkaisun sijaan. Merkittävämpi kysymys on, mikä alustariippumaton teknologia soveltuu kehitykseen parhaiten missäkin tilanteessa. Natiivisovellus on edelleen ja tulee todennäköisesti pysymään käytännön suorituskyvyssä alustariippumattomia ratkaisuja edellä vielä pitkään, mutta alustariippumattomien ohjelmistokehysten tarjoamat vaihtoehdot yltävät useimmissa tapauksissa jo nyt suorituskyvyltään riittävälle tasolle. Olennaista onkin pohtia mitä lisäarvoa natiivisti toteutettu sovellus kykenee tarjoamaan ollakseen perusteltu vaihtoehto alustariippumattomien ratkaisujen kustannustehokkuutta vastaan. Useimmiten alustariippumattomat teknologiat kykenevät täyttämään merkittävimmät sovellukselle asetetut vaatimukset, jolloin valinta tehdään natiivin ja alustariippumattoman sijaan hybridin ja hybridinatiivin välillä.

Käytännössä tehtäessä lopullista valintaa hybridin ja hybridinatiivin toteutustavan välillä merkittävä tekijä on tarjolla olevien kehittäjien kokemus kyseisistä toteutusteknologioista. Ammattitaitoiset kehittäjät kykenevät oppimaan uuden toteutusteknologian kohtalaisen lyhyessä ajassa, joten kysymykseksi nousee, onko toteutusteknologian opiskeluun käytetty aika kannattavaa. Riippuen toteutettavan sovelluksen koosta opiskeluun käytetyt resurssit saattavat olla hyvin pieni osa kokonaiskustannuksista, jolloin kyseisen ja tulevaisuuden projektien kannalta on uuden teknologian opiskeluun käytetty aika usein kannattava sijoitus. Mikäli toteutettava sovellus on kuitenkin pieni (n. 10 000 – 20 000€) ja opiskeltavan toteutusteknologian suosio tulevaisuudessa epävarma, on syytä harkita kehittäjille entuudestaan tuttua toteutusteknologiaa.

5.9 Kehitysideat

Alustariippumattomien ohjelmistokehysten toiminnan kannalta oleellista on suorituskyvyn yhteneväisyys kaikilla tuetuilla alustoilla. Tässä työssä tutkitut ohjelmistokehykset

tukevat Android-käyttöjärjestelmän lisäksi myös Apple iOS -käyttöjärjestelmää, minkä vuoksi vastaavat testit olisi hyödyllistä suorittaa myös iOS-alustalla. Alustojen erot vaikuttavat todennäköisesti lähinnä Ionic-ohjelmistokehyksen toimintaan, sillä sovellusta ajetaan alustan tarjoaman WebView-näkymän sisällä. React Nativen osalta sovelluksen käyttöliittymän käytännön suorituskyky tuskin eroaa natiivista vastaavasta, sillä sovelluksen komponentit ovat natiiveja. WebView-näkymien suorituskykyä on hankala verrata keskenään alustojen välillä, sillä suorituskykyyn vaikuttaa olennaisesti myös testilaitteen komponenttien suorituskyky. Koska sekä Android- että iOS-alustojen WebView-näkymät pohjautuvat samoihin teknologioihin, kuin työpöytäkäytössä olevat Chrome- ja Safari-selaimet, on suorituskykyä mahdollista arvioida suuntaa-antavasti näiden välillä. Apple iOS-alustan ja Safari-selaimen käyttämä Nitro-JavaScript-virtuaalikone on viime aikoina ollut Android-alustan käyttämää Cromium V8 -JavaScript-virtuaalikonetta tehokkaampi, joten tämän perusteella WebView-näkymä tarjoaa todennäköisesti parempaa suorituskykyä iOS-alustalla verrattuna tässä työssä tutkittuun Android-alustaan. [50]

Tässä työssä saadut tulokset kuvaavat tilannetta ainoastaan Android versiota 4.4 käyttävien tai tätä uudempien laitteiden tapauksessa. Mikäli toteutettavan sovelluksen pääasiallinen tai merkittävä kohdealusta on tätä vanhempi, eivät tulokset ole vertailukelpoisia. Vanhempi kohdealusta vaikuttaa erityisesti Ionic-testisovelluksen suorituskykyyn, sillä vanhemmat WebView-näkymät ovat suorituskyvyltään tässä työssä käytetyn testilaitteen WebView-näkymää huonommat. Testit on mahdollista suorittaa Ionic-testisovellukselle Android versiosta 4.0 ja iOS versiosta 6.0 alkaen. React Native -testisovellus puolestaan tukee Android versiota 4.1 ja iOS versiota 7.0 uudempia käyttöjärjestelmiä. Vastaavasti testit olisi aiheellista suorittaa uudemmilla laitteilla, kun Android-alustan Chromium-selaimeen pohjautuva WebView päivittyy testeissä käytettyä versiota 48.0.25641.106 uudemmaksi. WebView:n päivittyessä testisovellusten erot testisovellusten välillä todennäköisesti kapenevat Ionic-sovelluksen suorituskyvyn parantuessa.

Käytetty testilaite edusti kirjoitushetkellä tehoiltaan keskimääräistä parempitasoista laitetta. Keskeistä olisi myös testata testisovellusten toimivuutta pienitehoisemmilla laitteilla. Erityisesti Ionic-testisovelluksen osalta suoritinkäyttö oli odottamattoman korkea. Jotta sovelluskehysten toiminnasta olisi mahdollista saada laaja-alaisempi kuva, pitäisi testit suorittaa vielä matalan hintaluokan laitteilla, jolloin esimerkiksi suorittimen tarjoama teho saattaa loppua kesken vaikuttaen selkeästi sovelluksen suorituskykyyn ja mittaustuloksiin.

6. YHTEENVETO

Mobiililaitteilla toimivien palveluiden toteutukseen on yleisesti käytössä kolme mahdollista toteutustapaa: web-sovellus, natiivisovellus sekä hybridisovellus. Web-sovellus on käytännössä responsiivisesti toteutettu verkkosivusto, joka hyödyntää useimmiten SPA-arkkitehtuuria. Web-sovellus on useimmissa tapauksissa edullinen vaihtoehto, sillä laaja alustayhteensopivuus saavutetaan helposti jo yksittäiseen ohjelmakoodikantaan perustuvalla toteutuksella. Lisäksi responsiivisesti toteutettu web-sovellus mahdollistaa käytön myös työpöytälaitteilla ilman erillistä toteutusta. Ongelmana kuitenkin on, että web-sovellusta ei ole mahdollista asentaa markkinoiden suosituimpien alustojen laitteisiin ja web-sovelluksen käyttäminen vaatii laitteelta käytännössä aina internet-yhteyden.

Natiivisovellus on perinteisin tapa toteuttaa mobiililaitteisiin asennettavia sovelluksia. Natiivisti toteutetut sovellukset ovat edelleen vahva vaihtoehto, mikäli sovelluksen taustalogiikalta tai käyttöliittymältä vaaditaan ehdotonta suorituskykyä. Toteutustavan heikkous on sen hinta, mikäli sovellus on tarkoitus toteuttaa useammalle laitealustalle, sillä kutakin alustaa varten on kehitettävä oma toteutuksensa. Poikkeuksena kuitenkin on esimerkiksi React Native -ohjelmistokehyksen tarjoama vaihtoehto, jossa sovelluksen logiikka toteutetaan JavaScript-ohjelmointikielellä, ja ohjelmistokehys huolehtii oikeiden käyttöliittymäkomponenttien kutsumisesta, kunhan sovellus käännetään kullekin alustalle suoritettavaksi asennuspaketiksi. Kyseistä toteutustapaa kutsutaan yleisesti myös hybridinatiiviksi.

Hybridisovellukset yhdistävät web-sovellusten tarjoaman alustariippumattomuuden ja laajasti käytössä olevat web-kehityksessä käytetyt teknologiat ja paketoivat nämä laitteeseen asennettavaksi sovellukseksi. Hybridisovellus pohjautuu useimmiten HTML-merkkauskielellä toteutettuun käyttöliittymään ja JavaScript-ohjelmointikielellä kirjoitettuun taustalogiikkaan. Sovellus paketoidaan natiiviin sovellusrunkoon, joka käynnistyessään avaa WebView-näkymän, jolloin HTML-käyttöliittymä mallinnetaan internet-selaimen tapaan käyttäjälle näkyviksi elementeiksi. HTML-merkkauskieleen pohjautuvan käyttöliittymän ongelmana on heikompi suorituskyky, minkä vuoksi esimerkiksi animaatiot saattavat vaikuttaa kankeilta ja näkymien latautuminen saattaa kestää kauan verrattuna natiiviin tai hybridinatiiviin sovellukseen.

Perinteisesti alustakohtaisten natiivisovellusten kehittämistä on perusteltu suorituskyvyn näkökulmasta, mutta moderneilla alustariippumattomilla ohjelmistokehyksillä kehitetyt hybridit tai hybridinatiivit mobiilisovellukset tarjoavat nykypäivänä kilpailukykyisen vaihtoehdon useimmissa tilanteissa. Erityisesti hybridinatiivit ohjelmistokehykset, kuten React Native, kykenevät käyttöliittymänsä osalta useimmiten täysin natiivia vastaavaan suorituskykyyn. HTML-käyttöliittymään pohjautuvat hybridisovelluskehykset, kuten Ionic tarjoavat nopean tavan kehittää mobiilisovelluksia myös tilanteessa, jossa sovelluksen

kehittäjillä ei ole lainkaan kokemusta mobiilikehityksestä. Hybridisovellusten suorituskyky on riittävä suurimpaan osaan käyttötapauksista, mutta valtaosa käyttäjistä kykenee huomaamaan suorituskyvyn heikomman tason verrattuna natiivisovellukseen. Natiivisovelluksen toteutus saattaa olla edelleen perusteltua, mikäli sovelluksen käyttöliittymään kohdistuu erityisiä vaatimuksia suorituskyvyn ja ulkoasun näyttävyyden suhteen tai sovellukselta vaaditaan tukea laitteistolta asioihin, joihin alustariippumattomilta ohjelmistokehyksiltä ei löydy tukea.

Alustariippumattoman toteutustavan etuna on ennen kaikkea kustannustehokkuus kehitettäessä sovellusta useammalle laitealustalle, sillä sovellus tarvitsee optimitilanteessa toteuttaa vain kerran, ja ainoastaan kääntää halutuille alustoille yhteensopivaksi asennuspaketiksi. Joissain tapauksissa alustariippumattomalla ohjelmistokehyksellä toteutettu sovellus ei ole täysin alustariippumaton, vaan jonkin verran alustakohtaista toteutusta saatetaan vaatia. Alustariippumattomien ohjelmistokehysten keskeneräisyys asettaa myös haasteita sovelluksen kehitykselle, sillä yhdellä alustalla toimivaksi todettu sovellus saattaa olla kokonaan käyttökelvoton käännettäessä samaa sovellusta toiselle alustalle. Tämän vuoksi alustariippumatonta sovellusta kehitettäessä on tärkeää testata sovelluksen toimintaa aktiivisesti jokaisella kehitettävällä alustalla. Myös sovellusta jatkokehitettäessä ja otettaessa käyttöön uutta laitealustaa, on huomioitava, että sovellus ei välttämättä ole suoraan yhteensopiva uusien tai päivitettyjen alustojen kanssa.

Oikean toteutustavan valintaan vaikuttavat pääasiassa kehitysprojektiin saatavilla olevat resurssit ja sovelluksen kriittisimmät vaatimukset. Natiivisovellus soveltuu useimmiten tilanteisiin, joissa sovelluksen kalliimpi hinta muihin toteutustapoihin nähden ei ole este. Mikäli sovellukselta vaaditaan myös ehdottoman hyvää suorituskykyä tai laitteistolta vaaditaan ominaisuuksia, joita alustariippumattomat ohjelmistokehykset eivät tue, on natiivi myös perusteltu vaihtoehto. Mikäli tarjolla on runsaasti web-kehittäjiä, on hybridin tai hybridinatiivin sovelluksen toteuttaminen useimmiten edullista ja nopeaa. Hybridisovellusta voidaan myös nopeasta kehitysajastaan johtuen hyödyntää prototypointiin projektin alkuvaiheessa, vaikka lopullinen sovellus toteutettaisiin muuta tekniikkaa hyödyntäen. Hybridinatiivin etuna on käyttöliittymän natiivisovelluksen kaltainen suorituskyky, mutta merkittävänä puutteena esimerkiksi React Native -ohjelmistokehyksen tapauksessa ainoastaan kahden suosituimman mobiilikäyttöjärjestelmän tuki. Web-sovelluksen toteuttaminen soveltuu tilanteisiin, joissa sovelluksen asentaminen laitteeseen ei ole tärkeää ja mobiililaitteiden lisäksi halutaan tarjota sama palvelu myös työpöytäkäyttäjille. Web-sovellus tarjoaa useimmiten myös hyvän alustariippumattomuuden, sillä yhteensopivuus riippuu pääasiassa käytetystä selaimesta. Tapauksessa, jossa sovelluksesta on jo olemassa oleva web-sovellustoteutus, on lisäksi mahdollista toteuttaa laitteeseen sovelluskaupan kautta asennettava sovellus, jossa web-sovellusta ajetaan WebView-näkymän sisällä hybridisovelluksen kaltaisesti. Kyseisen toteutustavan avulla on mahdollista tarjota toimiva ratkaisu valtaosaan käyttötilanteista ja samanaikaisesti ratkaisu vaatii ainoastaan yksittäisen ohjelmakoodikannan ylläpitämistä.

LÄHTEET

[1] Android Developers, Application Fundamentals, 2015. Saatavissa (viitattu 29.12.2015): http://developer.android.com/guide/components/fundamentals.html

[2] Apache Cordova, Platform support, 2015. Saatavissa (viitattu 28.1.2016): https://cordova.apache.org/docs/en/latest/guide/support/index.html

[3] D. Albert, The Do's and Don'ts of Building HTML5 Hybrid Apps, 2014. Saatavissa (viitattu 9.1.2016): http://speckyboy.com/2014/12/18/dos-donts-building-html5-hybrid-apps/

[4] D. Aragon, Making Your App Work Offline: Tips and Cautionary Tales, 2014. Saatavissa (viitattu 6.1.2016): https://quickleft.com/blog/making-your-app-work-offline-tips-and-cautionary-tales/

[5] M. Asay. You're Using PhoneGap For All The Wrong Reasons, 2015. Saatavissa (viitattu 4.3.2016): http://readwrite.com/2015/06/24/phonegap-apache-cordova-cross-platform-tools

[6] A. Austin, Mobile App Developers are Suffering, 2015. Saatavissa (viitattu 2.11.2015): https://medium.com/swlh/mobile-app-developers-are-suffering-a5636c57d576

[7] M. Belshe et al, Hypertext Transfer Protocol Version 2, 2015. Saatavissa (viitattu 26.4.2016): https://http2.github.io/http2-spec/

[8] Apple Developer, About Objective-C, 2014. Saatavissa (viitattu 29.12.2015): https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[9] Apple Developer, App Store Review Guidelines, 2015. Saatavissa (viitattu 12.4.2016): https://developer.apple.com/app-store/review/guidelines/

[10] Apple Developer, iOS Human Interface Guidelines, Bars, 2016. Saatavissa (viitattu 28.2.2016): https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html

[11] Apple Developer, Swift. A modern programming language that is safe, fast, and interactive, 2016. Saatavissa (viitattu 26.4.2016): https://developer.apple.com/swift/

[12] A. Bradley, Where does the Ionic Framework fit in, 2013. Saatavissa (viitattu 9.1.2016): http://blog.ionic.io/where-does-the-ionic-framework-fit-in/

[13] C. Hampton, SASS: Syntatically Awesome Style Sheets, 2015. Saatavissa (viitattu 27.2.2016): http://sass-lang.com/

[14] CodeSchool, JavaScript, 2015. Saatavissa (viitattu 29.12.2015): https://www.javascript.com/

[15] Clutch, Cost to Build a Mobile App, 2015. Saatavissa (viitattu 6.1.2016): https://clutch.co/app-developers/resources/cost-build-mobile-app-survey

[16] Comentum, Native vs Hybrid / PhoneGap App Development Comparison, 2015. Saatavissa (viitattu 3.11.2015): http://www.comentum.com/phonegap-vs-native-app-development.html

[17] K. C. Dodds, Watching your AngularJS Watchers, 2014. Saatavissa (viitattu 9.1.2016): https://medium.com/@kentcdodds/counting-angularjs-watchers-11c5134dc2ef#.vff99qevd

[18] Drifty Co., Ionic Framework, 2015. Saatavissa (viitattu 29.12.2015): http://ionicframework.com/

[19] B. Eisenman, Learning React Native (O'Reilly), 2015. 254 s.

[20] Facebook Inc., Native UI Components, 2015. Saatavissa (viitattu 29.12.2015): https://facebook.github.io/react-native/docs/native-components-ios.html

[21] Facebook Inc., React, 2013. Saatavissa (viitattu 30.1.2016): https://facebook.github.io/react/index.html

[22] Facebook Inc., React Native, 2015. Saatavissa (viitattu 29.12.2015): https://facebook.github.io/react-native/

[23] P. Fischer, Build High-Performance HTML5 Cordova Apps with Crosswalk, 2015. Saatavissa (viitattu 15.1.2016): https://blogs.intel.com/evangelists/2015/05/12/build-high-performance-html5-cordova-apps-with-crosswalk/

[24] Formotus, Figuring the costs of custom mobile business app development, 2015. Saatavissa (viitattu 29.12.2015): http://www.formotus.com/14018/blog-mobility/figuring-the-costs-of-custom-mobile-business-app-development

[25] S. Giblisco, Write once, run anywhere (WORA), 2013. Saatavissa (viitattu 17.2.2016): http://whatis.techtarget.com/definition/write-once-run-anywhere-WORA

[26] Google, AngularJS, 2010.Saatavissa (viitattu 9.1.2016): https://angularjs.org/

[27] Google, AngularJS tutorial: Two-way Data Binding, 2010. Saatavissa (viitattu 9.1.2016): https://docs.angularjs.org/tutorial/step_04

[28] Google, Google Design, Structure, 2016. Saatavissa (viitattu 28.2.2016): https://www.google.com/design/spec/layout/structure.html

[29] Google, Search Console Help: Googlebot, 2016. Saatavissa (viitattu 4.3.2016): https://support.google.com/webmasters/answer/182072?hl=en

[30] Google Developers, Android Debug Bridge, 2015. Saatavissa (viitattu 20.3.2016): http://developer.android.com/tools/help/adb.html

[31] Google Developers, App indexing, 2015. Saatavissa (viitattu 4.3.2016): https://developers.google.com/app-indexing/

[32] C. Göllner, Google Play: System Monitor, 2016. Saatavissa (viitattu 18.3.2016): https://play.google.com/store/apps/details?id=com.cgollner.systemmonitor

[33] Intel Open Source Technology Center, Crosswalk, 2013. Saatavissa (viitattu 15.1.2016): https://crosswalk-project.org/

[34] International Data Corporation, Smartphone OS Market Share, 2015 Q2, 2015. Saatavissa (viitattu: 2.11.2015): http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[35] Ionic Forum, Crosswalk's performance, 2015. Saatavissa (viitattu 19.3.2016): https://forum.ionicframework.com/t/crosswalks-performance/16245

[36] A. Kinsey, Can Website Load Time Effect Your Business Income, 2015. Saatavissa (viitattu 28.2.2016): https://www.seoandy.net/optimisation/website-load-time/

[37] P. Koch, List of Android WebViews, 2015. Saatavissa (viitattu 26.3.2016): http://www.quirksmode.org/blog/archives/2015/02/android_webview.html

[38] KodersLab, React Nativer for Web, 2015. Saatavissa (viitattu 29.12.2015): https://github.com/KodersLab/react-native-for-web

[39] B. Leroux, PhoneGap, Cordova, and what's in a name, 2012. Saatavissa (viitattu 4.3.2016): http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/

[40] Microsoft, Choosing a Programming Language for Windows Mobile Development, 2010. Saatavissa (viitattu 29.12.2015): https://msdn.microsoft.com/en-us/library/bb677133.aspx

[41] Microsoft, Visual Studio Languages, 2016. Saatavissa (viitattu 6.1.2016): https://msdn.microsoft.com/en-us/vstudio/jj672990.aspx

[42] C. Mills, Hosted Apps, 2015. Saatavissa (viitattu 28.1.2016): https://developer.mozilla.org/en-US/Marketplace/Options/Hosted_apps

[43] M. Prakash, AngularJS – ngRepeat Performance Watchers, 2015. Saatavissa (viitattu 27.2.2016): http://excellencenodejsblog.com/angularjs-ngrepeat-performance-watchers/

[44] T. Roes, Old WebView vs. Chromium backed WebView Benchmark, 2013. Saatavissa (viitattu 26.3.2016): https://www.timroes.de/2013/11/23/old-webview-vs-chromium-webview/

[45] A. Singhal, Using site speed in web search ranking, 2010. Saatavissa (viitattu: 9.1.2016): https://googlewebmastercentral.blogspot.fi/2010/04/using-site-speed-in-web-search-ranking.html

[46] M. Spinelli, Performance tricks for (mobile) web development, 2014. Saatavissa (viitattu 28.4.2016): http://cubiq.org/performance-tricks-for-mobile-web-development

[47] Tizen, HTML5 Features on Tizen. Saatavissa (viitattu 28.1.2016): https://developer.tizen.org/community/tip-tech/html5-features-on-tizen

[48] T. VanToll, Why iOS 8's WKWebView is a Big Deal for Hybrid Development, 2014. Saatavissa (viitattu 26.3.2016): http://developer.telerik.com/featured/why-ios-8s-wkwebview-is-a-big-deal-for-hybrid-development/

[49] M. Viskari, Tilaisitko yhden sivun web-sovelluksen (SPA) vai arkkitehtuuriltaan valmiiksi vanhentuneen järjestelmän, 2015. Saatavissa (viitattu 3.4.2016): http://www.eatech.fi/fi/blogi/spa-sovellukset

[50] B. Widder, Battle of the best browsers, 2015. Saatavissa (viitattu 27.3.2016): http://www.digitaltrends.com/computing/best-browser-internet-explorer-vs-chrome-vs-firefox-vs-safari-vs-edge/

[51] D. Witte, React Native for Android: How we built the first cross-platform React Native app, 2015. Saatavissa (viitattu 17.2.2016): https://code.facebook.com/posts/1189117404435352/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/

[52] S. Work, How Loading Time Affects Your Bottom Line, 2011. Saatavissa (viitattu: 6.1.2016): https://blog.kissmetrics.com/loading-time/

[53] Xamarin, Building Cross Platform Applications Overview, 2015. Saatavissa (viitattu 29.12.2015): https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_0_-_overview/

[54] Xamarin, User Interface, 2015. Saatavissa (viitattu 29.12.2015): https://developer.xamarin.com/guides/android/user_interface/

LIITE A: KÄYTTÄJÄTESTIEN KYSYMYKSET

1. Mitä ajatuksia sovelluksen käytöstä tulee mieleen?
2. Minkä osa-alueen sovelluksesta koit toimivan erityisen hyvin?
3. Minkä osa-alueen sovelluksesta koit toimivan erityisen huonosti?
4. Poikkeaako sovelluksen toiminta jollain tapaa muista käyttämistäsi sovelluksista?
    - Kyllä: Millä tavoin sovelluksen toiminta on poikkeavaa?
    - Ei: Millä tavoin sovelluksen toiminta muistuttaa muita käyttämiäsi sovelluksia?
5. Mitä parantaisit sovelluksen toiminnassa?

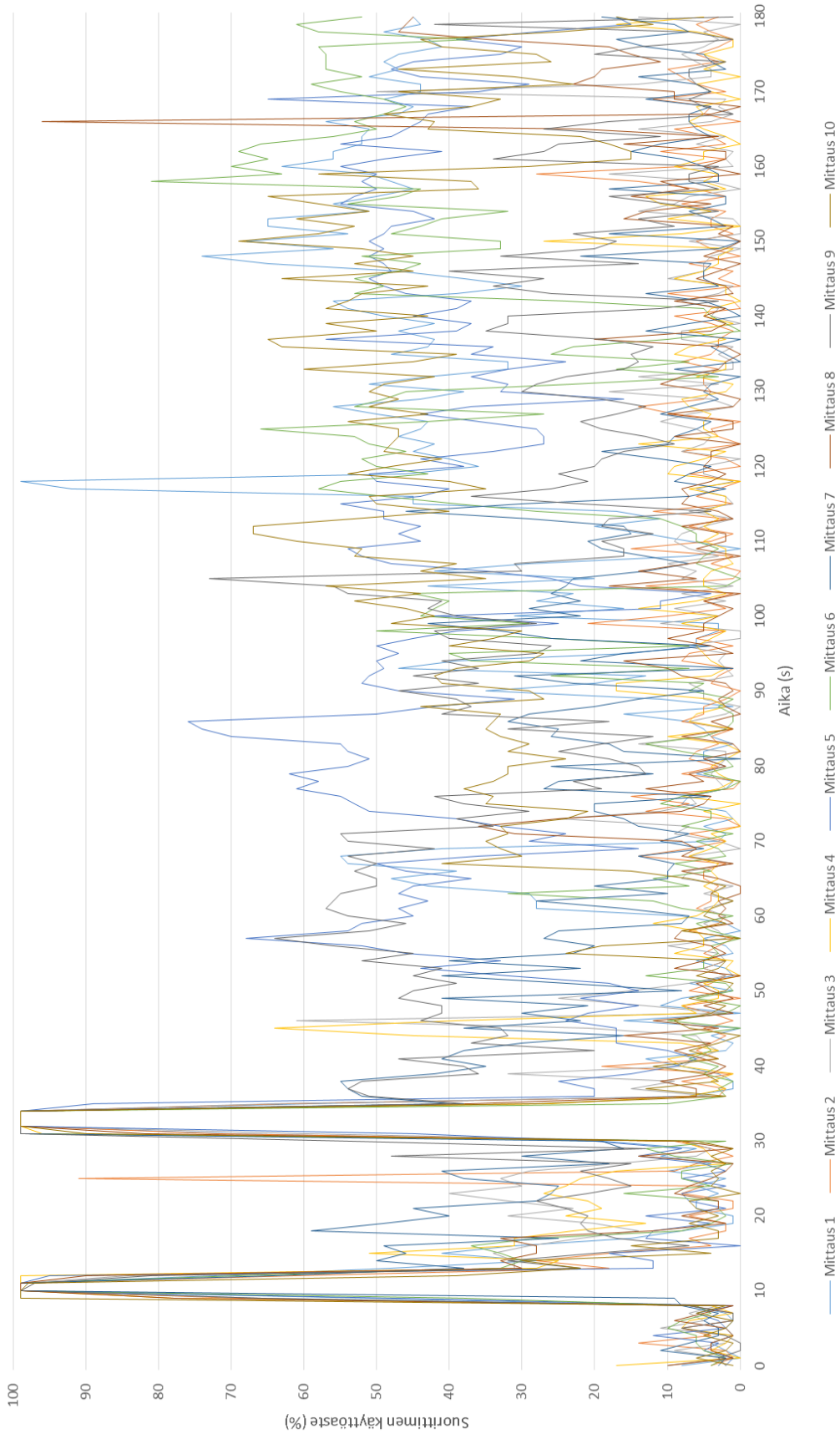
LIITE B: IONIC TESTISOVELLUKSEN SUORITTIMEN KÄYTTÖ-ASTE: MITTAUSTULOKSET

Aika (s)	Mittaus 1	Mittaus 2	Mittaus 3	Mittaus 4	Mittaus 5	Mittaus 6	Mittaus 7	Mittaus 8	Mittaus 9	Mittaus 10	Kes-kiarvo	Keskiha-jonta
0	8	4	10	17	2	4	3	10	2	1	6,10	5,07
1	0	1	2	0	1	2	2	1	3	6	1,80	1,75
2	2	2	9	4	5	5	11	4	0	2	4,40	3,37
3	4	14	2	2	3	6	7	4	0	1	4,30	4,03
4	1	4	1	1	12	6	3	1	2	5	3,60	3,47
5	2	8	11	8	2	10	3	2	8	2	5,60	3,72
6	1	1	5	7	3	8	5	9	1	1	4,10	3,14
7	2	1	7	1	6	4	3	5	1	6	3,60	2,32
8	7	7	3	4	1	6	8	1	7	2	4,60	2,72
9	51	69	66	78	48	38	9	78	66	99	60,20	25,01
10	99	99	99	99	99	99	99	99	99	99	99,00	0,00
11	99	99	99	99	99	99	99	99	97	98	98,70	0,67
12	72	61	84	99	95	69	64	90	81	39	75,40	18,18
13	48	18	31	28	12	33	38	30	22	22	28,20	10,36
14	28	28	31	25	12	30	50	33	32	30	29,90	9,26
15	41	18	34	51	18	33	46	28	20	4	29,30	14,55
16	31	4	29	31	0	37	49	28	7	15	23,10	15,88
17	15	7	22	31	13	28	25	33	17	3	19,40	10,11
18	8	2	14	23	12	8	59	12	21	3	16,20	16,50
19	1	2	20	13	4	4	49	2	22	7	12,40	14,93
20	1	7	32	24	13	2	40	8	21	3	15,10	13,57
21	6	1	23	19	2	3	45	3	24	6	13,20	14,31
22	6	1	32	21	7	3	28	3	28	7	13,60	12,19
23	3	8	40	27	8	16	26	9	21	0	15,80	12,58
24	6	5	30	24	2	4	25	7	15	4	12,20	10,45
25	2	91	33	22	8	8	38	4	18	3	22,70	27,13
26	10	9	29	17	2	8	41	2	22	4	14,40	12,87
27	4	6	17	2	5	1	18	1	15	2	7,10	6,84
28	11	1	6	4	14	9	30	14	48	4	14,10	14,45
29	6	4	4	5	8	13	16	1	13	1	7,10	5,26
30	19	9	8	8	20	2	19	8	54	6	15,30	14,94
31	70	72	60	96	45	99	99	82	99	90	81,20	18,91
32	99	99	99	99	99	99	99	99	99	99	99,00	0,00
33	99	99	99	99	99	99	99	99	99	99	99,00	0,00
34	99	99	99	99	99	99	99	99	99	99	99,00	0,00
35	59	62	64	26	89	10	40	37	38	28	45,30	23,08
36	7	3	4	7	20	2	52	6	51	2	15,40	19,72
37	1	2	15	2	20	3	54	6	54	13	17,00	20,52
38	1	8	9	4	25	1	55	11	52	3	16,90	20,52
39	3	1	32	1	15	5	42	4	36	5	14,40	16,03
40	8	19	8	8	9	3	35	2	38	12	14,20	12,66
41	13	3	6	10	6	7	41	5	47	3	14,10	16,11
42	2	6	3	3	10	4	38	9	20	7	10,20	11,09
43	1	4	5	8	17	5	30	4	37	3	11,40	12,54
44	4	2	6	45	17	8	16	16	32	0	14,60	14,29
45	0	5	21	64	17	0	38	3	33	7	18,80	20,84
46	16	1	61	39	24	10	22	12	44	9	23,80	18,72
47	0	4	2	6	21	2	30	1	41	4	11,10	14,40
48	11	8	13	8	14	0	21	5	41	2	12,30	11,80
49	8	0	25	3	22	6	41	1	47	7	16,00	16,99
50	1	7	15	3	14	1	8	4	45	2	10,00	13,29
51	7	4	7	5	18	2	24	6	39	1	11,30	12,13
52	1	2	2	0	32	13	41	0	45	6	14,20	18,04
53	2	5	5	2	44	5	22	9	41	3	13,80	16,23
54	6	2	2	1	33	5	40	5	52	2	14,80	19,15
55	1	5	3	9	46	3	23	2	45	24	16,10	17,58
56	2	3	10	5	52	4	20	4	56	19	17,50	20,30
57	5	1	2	5	68	0	27	9	64	0	18,10	26,48
58	0	7	3	6	54	4	25	8	51	8	16,60	20,07
59	2	3	5	12	52	6	8	1	46	2	13,70	18,94
60	8	2	2	6	45	1	7	3	54	5	13,30	19,33
61	28	6	5	4	47	8	16	2	57	4	17,70	19,79
62	28	4	1	3	43	12	28	3	56	1	17,90	19,80
63	29	4	5	3	47	32	10	0	55	4	18,90	20,26
64	43	3	1	5	45	7	20	0	50	2	17,60	20,45
65	48	5	8	3	37	12	10	3	50	7	18,30	18,93
66	39	5	7	4	46	4	10	8	53	15	19,10	19,13
67	54	2	1	3	50	9	9	1	50	41	22,00	23,41
68	55	14	7	7	36	2	14	8	54	30	22,70	19,84
69	41	7	0	4	14	3	5	6	42	33	15,50	16,54
70	3	2	3	2	29	6	11	8	54	35	15,30	17,91
71	2	7	9	4	24	1	7	31	55	32	17,20	17,73
72	5	0	7	0	33	8	14	36	34	33	17,00	15,18

73	1	2	25	1	39	4	16	25	39	24	17,60	15,09
74	2	13	11	5	51	4	20	8	29	21	16,40	14,88
75	8	7	6	0	53	11	20	5	38	35	18,30	17,71
76	7	15	7	8	55	8	4	4	42	34	18,40	18,39
77	2	1	2	3	61	2	27	13	19	38	16,80	20,07
78	1	0	3	1	58	0	25	5	23	34	15,00	19,61
79	6	8	4	3	62	5	12	7	13	32	15,20	18,47
80	2	3	3	4	54	1	26	2	14	32	14,10	17,85
81	5	7	2	2	51	2	0	5	18	24	11,60	15,87
82	5	6	2	0	54	8	16	0	25	32	14,80	17,51
83	1	3	14	2	55	13	18	1	19	29	15,50	16,76
84	4	4	5	10	70	5	26	8	12	33	17,70	20,88
85	5	1	2	1	74	2	25	1	32	35	17,80	24,08
86	7	8	7	7	76	1	32	4	18	34	19,40	22,96
87	16	5	2	6	50	1	29	0	41	33	18,30	18,51
88	1	5	0	2	42	3	20	2	37	44	15,60	18,49
89	11	2	5	4	31	1	14	3	39	27	13,70	13,77
90	35	0	5	17	47	7	5	1	47	29	19,30	18,70
91	20	4	7	17	52	5	23	2	36	41	20,70	17,35
92	13	3	1	4	51	26	31	8	45	42	22,40	19,06
93	47	1	2	2	49	7	1	10	36	39	19,40	20,61
94	39	8	3	4	50	37	22	16	41	29	24,90	16,80
95	15	7	2	5	47	40	16	1	28	27	18,80	16,15
96	6	1	8	6	50	7	4	4	26	40	15,20	17,27
97	6	6	0	4	45	26	26	10	40	34	19,70	16,47
98	3	2	0	2	39	50	33	5	42	30	20,60	19,92
99	3	21	11	8	25	29	43	5	28	48	22,10	15,56
100	31	10	3	3	42	44	22	3	39	42	23,90	17,78
101	16	2	9	14	11	42	29	1	43	46	21,30	17,30
102	28	5	2	6	11	40	22	5	41	53	21,30	18,35
103	23	1	8	2	4	45	26	0	54	44	20,70	20,77
104	43	13	1	5	22	2	24	18	56	57	24,10	21,11
105	20	1	8	5	26	0	23	6	73	35	19,70	22,12
106	42	0	6	4	35	3	13	14	30	44	19,10	17,01
107	29	4	10	1	48	6	7	7	31	39	18,20	16,88
108	7	1	3	2	52	5	13	0	16	53	15,20	20,32
109	0	15	7	2	54	3	19	6	16	52	17,40	19,81
110	7	4	9	5	44	6	21	2	19	61	17,80	19,72
111	13	1	8	3	47	6	15	2	12	67	17,40	21,91
112	20	7	5	5	44	8	16	8	19	67	19,90	20,30
113	11	2	1	3	49	11	29	1	18	53	17,80	19,62
114	17	12	7	5	49	28	46	3	4	40	21,10	18,20
115	45	1	1	2	55	39	30	8	22	50	25,30	21,41
116	45	6	2	2	44	49	7	7	37	51	25,00	21,66
117	92	3	6	7	40	58	2	8	26	35	27,70	29,47
118	99	0	1	0	50	55	9	4	21	40	27,90	32,80
119	46	7	5	10	51	43	7	2	25	54	25,00	21,30
120	36	0	5	9	38	50	4	5	20	48	21,50	19,63
121	40	2	0	5	44	52	11	4	19	41	21,80	20,26
122	45	8	8	2	34	46	19	4	16	49	23,10	18,64
123	42	0	5	14	27	51	10	0	9	48	20,60	19,88
124	47	3	4	3	27	53	9	9	13	47	21,50	20,27
125	44	4	6	5	28	66	6	1	19	47	22,60	22,74
126	43	0	11	5	36	43	4	1	22	54	21,90	20,44
127	50	9	7	4	44	27	11	10	17	43	22,20	17,45
128	56	14	1	7	37	53	7	1	13	51	24,00	22,66
129	44	5	4	8	16	49	3	0	19	47	19,50	19,65
130	38	4	18	2	33	46	5	4	30	51	23,10	18,85
131	51	5	3	1	32	23	4	11	28	49	20,70	18,93
132	43	5	14	6	37	3	0	8	24	42	18,20	16,92
133	32	3	1	2	32	17	9	3	16	60	17,50	18,96
134	32	8	1	2	24	7	0	2	14	45	13,50	15,35
135	48	4	3	9	37	26	2	0	15	39	18,30	17,81
136	43	3	1	7	34	23	4	2	12	63	19,20	21,19
137	42	3	8	2	57	10	0	20	19	65	22,60	23,73
138	47	1	8	7	39	0	13	4	35	50	20,40	19,98
139	42	4	0	1	37	2	7	2	32	57	18,40	21,33
140	49	9	4	5	45	4	0	5	32	43	19,60	20,06
141	54	0	3	4	39	5	2	2	11	57	17,70	22,93
142	56	1	8	0	37	24	5	9	3	54	19,70	21,85
143	39	10	3	2	46	53	13	1	26	52	24,50	21,36
144	30	0	1	2	49	49	3	2	34	43	21,30	21,57
145	38	3	10	9	51	53	5	6	27	63	26,50	23,05
146	47	1	5	5	48	47	5	3	40	45	24,60	22,06
147	65	7	5	3	49	44	4	0	14	53	24,40	25,19
148	74	0	4	3	51	52	22	5	33	45	28,90	25,98
149	56	4	0	1	49	33	3	2	20	52	22,00	23,38
150	68	6	7	27	51	33	0	0	17	69	27,80	26,77
151	54	1	0	8	49	48	18	3	23	58	26,20	23,68
152	65	4	0	0	48	44	2	1	9	53	22,60	26,39
153	65	4	1	10	42	41	4	16	14	61	25,80	24,31
154	51	2	14	4	45	32	7	13	3	51	22,20	20,45

155	56	8	9	11	55	54	2	4	10	58	26,70	25,16
156	50	2	7	13	53	47	2	15	18	65	27,20	23,85
157	45	6	0	2	50	44	18	7	3	36	21,10	20,35
158	48	14	4	5	52	81	3	11	7	37	26,20	26,85
159	51	28	18	3	50	63	3	0	7	58	28,10	25,19
160	63	8	1	9	55	70	5	5	3	29	24,80	27,47
161	56	2	2	5	49	65	8	2	34	15	23,80	24,90
162	56	11	1	5	41	69	15	2	27	15	24,20	23,69
163	52	2	6	0	55	66	11	16	25	18	25,10	23,92
164	52	4	2	2	48	56	4	3	11	22	20,40	22,70
165	51	9	14	5	46	50	6	22	27	43	27,30	18,75
166	57	0	4	7	44	53	7	96	18	42	32,80	31,01
167	47	5	1	5	43	49	7	4	1	49	21,10	22,42
168	45	4	3	6	37	46	4	0	5	37	18,70	19,68
169	49	12	2	0	65	49	13	9	7	33	23,90	23,22
170	44	2	50	4	36	55	4	9	4	47	25,50	22,60
171	44	6	14	6	29	59	6	23	6	23	21,60	18,23
172	51	8	4	0	44	52	14	20	7	31	23,10	19,97
173	47	10	4	5	48	57	2	19	7	47	24,60	22,30
174	49	3	6	2	45	57	4	11	2	26	20,50	21,96
175	47	6	8	5	33	57	5	15	20	28	22,40	18,52
176	41	3	5	1	30	58	13	18	14	41	22,40	19,24
177	43	1	3	1	39	37	17	35	1	44	22,10	19,20
178	49	4	5	6	28	58	7	47	7	25	23,60	21,05
179	44	6	0	17	15	61	9	46	42	12	25,20	20,98
180	45	3	14	5	17	52	19	45	1	17	21,80	18,77

LIITE C: IONIC-TESTISOVELLUKSEN SUORITTIMEN KÄYTTÖASTE: KUVAAJAT MITTAUSJAKSOITTAIN



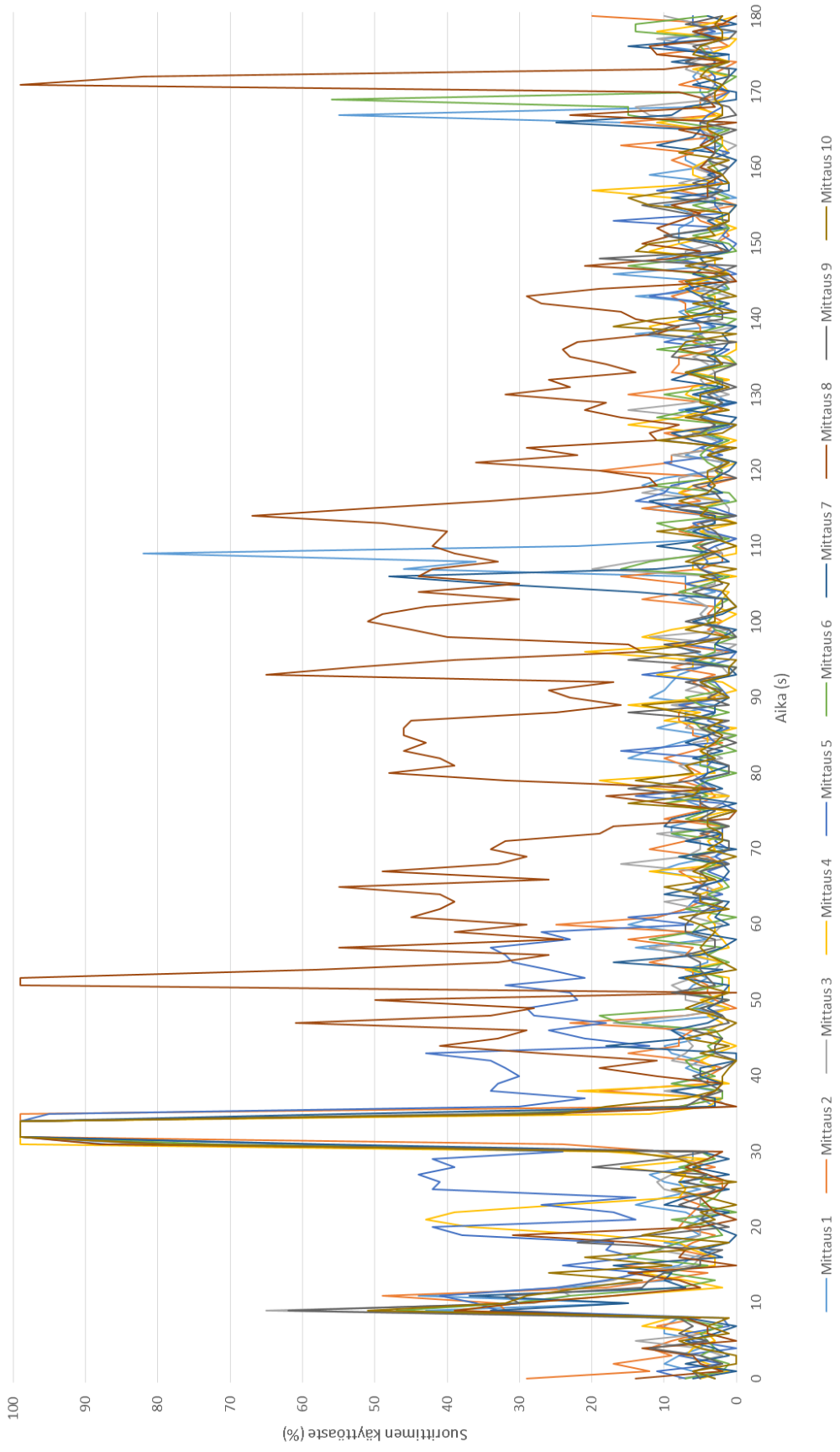
LIITE D: REACT NATIVE -TESTISOVELLUKSEN SUORITTIMEN KÄYTTÖASTE: MITTAUSTULOKSET

Aika (s)	Mittaus 1	Mittaus 2	Mittaus 3	Mittaus 4	Mittaus 5	Mittaus 6	Mittaus 7	Mittaus 8	Mittaus 9	Mittaus 10	Keskiarvo	Keskihajonta
0	3	29	8	5	7	7	6	14	5	3	8,70	7,79
1	6	12	1	1	11	2	0	2	6	6	4,70	4,24
2	10	17	7	4	2	1	7	5	2	0	5,50	5,10
3	8	9	2	7	8	5	1	6	5	0	5,10	3,14
4	5	12	5	6	0	3	3	13	12	5	6,40	4,43
5	4	9	14	3	6	4	7	0	4	8	5,90	3,87
6	10	5	4	4	7	5	3	4	8	1	5,10	2,60
7	10	11	4	13	1	2	0	4	6	4	5,50	4,43
8	8	5	8	9	2	9	7	7	8	1	6,40	2,84
9	43	32	65	39	33	47	34	39	62	51	44,50	11,72
10	15	33	28	30	36	31	15	30	30	32	28,00	7,18
11	44	49	23	17	41	23	37	17	32	29	31,20	11,32
12	24	18	25	2	25	8	7	5	13	21	14,80	8,89
13	9	11	17	7	18	3	10	8	12	13	10,80	4,52
14	14	4	14	13	13	10	9	15	10	26	12,80	5,67
15	8	16	6	3	24	4	17	0	5	9	9,20	7,50
16	6	3	5	3	14	15	2	8	5	21	8,20	6,34
17	3	8	11	6	18	11	5	7	2	7	7,80	4,64
18	10	6	8	8	17	8	1	14	22	1	9,50	6,64
19	5	3	5	19	38	2	0	31	12	13	12,80	12,93
20	7	8	5	37	42	3	3	5	1	7	11,80	14,80
21	5	7	8	43	14	9	5	0	5	3	9,90	12,22
22	7	6	3	39	17	0	1	3	4	5	8,50	11,70
23	14	5	5	24	27	5	10	4	8	0	10,20	8,92
24	10	8	5	8	14	3	7	5	5	4	6,90	3,28
25	5	2	10	7	42	2	1	2	3	9	8,30	12,26
26	10	2	11	8	41	5	8	1	7	0	9,30	11,76
27	12	9	9	7	44	4	1	4	4	9	10,30	12,29
28	7	3	5	16	39	8	6	7	20	5	11,60	11,00
29	6	8	7	4	42	3	1	4	10	8	9,30	11,80
30	6	10	11	18	24	5	5	2	4	15	10,00	7,09
31	87	24	81	99	60	72	57	88	78	79	72,50	21,20
32	99	99	99	99	99	99	99	99	99	99	99,00	0,00
33	99	99	99	99	99	99	99	99	99	99	99,00	0,00
34	99	99	99	99	99	99	99	99	99	99	99,00	0,00
35	29	99	39	12	95	32	54	27	30	24	44,10	29,87
36	3	8	1	4	30	8	3	0	7	18	8,20	9,21
37	5	2	3	2	21	2	3	5	5	7	5,50	5,70
38	3	19	10	22	34	2	9	3	2	3	10,70	10,91
39	9	4	2	1	33	9	2	2	4	2	6,80	9,65
40	4	4	5	5	30	3	5	11	6	2	7,50	8,26
41	7	2	3	3	32	6	5	19	2	1	8,00	9,90
42	5	6	9	5	34	7	0	11	0	0	7,70	10,00
43	9	15	7	3	43	3	0	27	3	5	11,50	13,59
44	10	8	6	0	12	4	18	41	3	1	10,30	12,08
45	5	8	7	3	21	2	7	33	2	7	9,50	9,89
46	9	6	3	4	26	3	9	29	6	2	9,70	9,71
47	13	23	17	2	18	16	4	61	1	0	15,50	17,97
48	2	7	7	4	28	19	2	34	2	3	10,80	11,88
49	6	0	1	1	29	5	4	28	5	7	8,60	10,74
50	3	3	7	5	22	1	2	50	1	4	9,80	15,41
51	7	4	7	1	23	7	2	79	9	1	6,78	6,76
52	6	4	9	1	32	2	1	99	3	5	16,20	30,50
53	4	6	6	1	21	2	8	99	4	7	15,80	29,75
54	7	6	3	7	26	6	2	58	5	0	12,00	17,66
55	4	12	2	2	31	1	17	33	5	2	10,90	12,24
56	4	7	7	6	32	4	5	26	7	4	10,20	10,09
57	14	6	12	1	34	2	2	55	3	5	13,40	17,65
58	4	15	2	2	23	12	0	24	3	4	8,90	9,01
59	8	6	3	3	27	6	6	39	7	3	10,80	12,16
60	15	25	2	3	6	5	1	29	7	5	9,80	9,89
61	11	11	6	4	15	0	3	45	6	8	10,90	12,76
62	7	7	5	3	2	7	2	41	4	1	7,90	11,85
63	6	5	10	5	6	2	1	39	2	5	8,10	11,16
64	5	4	2	5	2	4	10	41	6	5	8,40	11,67
65	2	4	4	2	6	1	5	55	5	10	9,40	16,22
66	3	6	5	3	1	2	2	26	4	3	5,50	7,35
67	11	8	5	12	4	5	1	49	5	6	10,60	13,88
68	8	2	16	1	1	3	2	33	1	4	7,10	10,22
69	3	4	7	6	3	8	8	29	6	0	7,40	8,00
70	2	12	5	3	1	1	0	34	1	4	6,30	10,33
71	6	7	5	4	3	1	7	32	1	2	6,80	9,14
72	9	4	11	2	2	9	2	19	5	2	6,50	5,56

73	9	2	1	2	2	1	10	17	1	4	4,90	5,38
74	5	10	4	6	4	7	9	1	5	4	5,50	2,64
75	3	5	4	1	0	4	4	0	3	0	2,40	1,96
76	6	5	2	10	2	10	0	10	5	15	6,50	4,67
77	9	3	1	1	14	4	7	18	5	6	6,80	5,53
78	3	4	4	11	4	6	2	3	15	5	5,70	4,11
79	5	8	5	19	6	4	5	32	6	14	10,40	8,98
80	1	5	6	3	4	0	4	48	1	6	7,80	14,28
81	4	7	8	4	4	5	1	39	1	7	8,00	11,15
82	15	10	2	4	3	5	6	41	4	3	9,30	11,81
83	12	6	4	5	16	0	2	46	4	5	10,00	13,49
84	8	2	6	4	3	3	7	43	0	3	7,90	12,56
85	4	6	0	4	4	0	4	46	3	2	7,30	13,73
86	11	6	7	0	1	5	2	46	5	4	8,70	13,48
87	7	8	6	10	4	6	7	45	1	2	9,60	12,73
88	7	8	3	5	3	1	3	25	15	8	7,80	7,24
89	3	7	9	15	8	5	3	16	1	13	8,00	5,25
90	12	3	1	4	1	7	1	23	4	1	5,70	7,01
91	10	4	7	0	2	2	3	26	2	3	5,90	7,62
92	9	6	4	2	3	4	5	17	7	5	6,20	4,29
93	8	3	1	11	13	1	0	65	1	1	10,40	19,75
94	5	9	2	6	5	0	1	53	0	1	8,20	16,02
95	3	5	8	6	1	6	7	39	15	0	9,00	11,33
96	6	1	2	21	0	3	0	13	5	14	6,50	7,14
97	9	9	0	6	7	3	3	15	10	6	6,80	4,26
98	2	1	12	13	4	0	2	40	0	1	7,50	12,37
99	1	4	1	9	3	3	0	45	1	7	7,40	13,52
100	4	2	4	2	4	4	6	51	7	3	8,70	14,94
101	5	4	5	0	3	3	2	49	3	3	7,70	14,58
102	4	3	4	3	3	3	0	43	2	0	6,50	12,90
103	8	13	6	2	3	1	1	30	2	1	6,70	9,07
104	3	7	8	1	2	8	14	44	6	2	9,50	12,74
105	7	7	3	7	7	4	34	30	2	1	10,20	11,75
106	7	16	3	0	2	1	48	44	3	5	12,90	18,04
107	46	5	20	6	5	16	11	42	5	0	15,60	16,11
108	36	2	14	6	4	12	1	33	6	7	12,10	12,49
109	82	2	2	0	5	4	3	39	3	4	14,40	26,36
110	22	6	3	0	3	6	11	42	5	0	9,80	13,01
111	2	4	0	3	0	1	3	41	2	3	5,90	12,40
112	7	8	5	5	4	7	6	40	3	11	9,60	10,92
113	3	5	6	0	6	11	3	49	5	0	8,80	14,48
114	4	3	1	3	0	1	3	67	0	5	8,70	20,55
115	13	13	1	5	3	5	9	51	9	4	11,30	14,53
116	1	5	2	2	14	0	12	34	2	8	8,00	10,32
117	9	8	13	8	10	1	2	19	3	4	7,70	5,54
118	13	4	8	6	9	12	4	11	6	3	7,60	3,57
119	10	0	8	2	4	3	3	12	0	4	4,60	4,09
120	3	19	3	3	6	2	1	19	3	4	6,30	6,82
121	2	9	1	0	10	4	0	36	1	1	6,40	11,01
122	7	9	9	4	2	5	4	22	4	4	7,00	5,75
123	5	2	1	3	5	4	1	29	1	0	5,10	8,58
124	2	6	3	0	2	1	6	11	10	11	5,20	4,24
125	6	10	6	4	4	5	9	12	5	2	6,30	3,09
126	2	1	1	15	0	1	1	8	3	0	3,20	4,76
127	4	7	5	8	5	11	0	16	5	7	6,80	4,32
128	8	2	15	4	4	6	7	21	4	1	7,20	6,23
129	1	4	6	4	0	3	0	18	2	5	4,30	5,23
130	3	15	1	7	6	10	6	32	4	5	8,90	9,00
131	5	7	4	6	2	3	2	23	0	1	5,30	6,60
132	2	1	5	1	2	2	9	26	3	4	5,50	7,59
133	7	9	3	6	7	7	6	14	3	7	6,90	3,11
134	3	8	4	0	2	0	2	18	0	2	3,90	5,51
135	3	8	6	2	3	1	4	23	9	3	6,20	6,44
136	3	3	3	0	1	11	2	24	8	8	6,30	7,15
137	5	6	4	0	10	3	1	22	0	7	5,80	6,53
138	14	5	5	11	5	6	2	11	12	0	7,10	4,63
139	3	5	0	12	3	2	0	8	9	17	5,90	5,55
140	6	6	4	7	6	0	8	14	2	11	6,40	4,06
141	8	7	3	4	0	7	4	16	2	0	5,10	4,75
142	4	7	4	3	3	2	1	27	4	6	6,10	7,55
143	14	9	1	3	12	4	5	29	5	0	8,20	8,60
144	1	5	5	8	6	1	7	19	3	3	5,80	5,20
145	4	8	0	5	7	2	2	0	1	7	3,60	3,03
146	17	1	3	1	0	5	6	1	3	1	3,80	5,03
147	3	4	6	3	3	15	4	21	0	7	6,60	6,45
148	11	9	7	3	9	6	5	9	19	2	8,00	4,81
149	5	2	7	12	1	0	1	5	1	14	4,80	4,89
150	8	3	3	8	0	1	2	13	4	12	5,40	4,58
151	8	1	2	3	2	6	10	9	10	3	5,40	3,60
152	8	2	1	0	1	1	2	11	3	4	3,30	3,53
153	6	5	3	3	17	2	4	8	1	1	5,00	4,76
154	6	6	7	1	2	2	1	5	7	1	3,80	2,62

155	10	0	2	0	3	6	0	9	13	12	5,50	5,17
156	0	3	5	9	3	1	9	4	4	15	5,30	4,50
157	3	8	2	20	11	4	1	4	2	8	6,30	5,79
158	2	2	8	3	3	4	1	4	6	1	3,40	2,22
159	12	3	3	6	3	2	4	2	2	2	3,90	3,11
160	5	6	6	6	5	3	1	5	1	4	4,20	1,93
161	7	9	1	4	4	4	0	4	4	1	3,80	2,74
162	7	6	2	1	1	3	3	3	3	8	3,70	2,45
163	2	16	0	2	5	2	11	5	3	4	5,00	4,88
164	5	3	3	2	4	4	6	3	3	2	3,50	1,27
165	2	4	5	4	5	1	5	8	0	4	3,80	2,30
166	16	16	2	11	5	8	25	0	5	4	9,20	7,79
167	55	3	12	2	4	15	9	23	0	2	12,50	16,58
168	5	5	14	6	2	15	7	3	1	2	6,00	4,88
169	4	5	3	2	3	56	0	4	5	3	8,50	16,75
170	5	1	4	1	1	4	0	8	3	3	3,00	2,40
171	5	8	3	2	3	2	6	99	2	2	13,20	30,22
172	6	4	4	1	3	0	1	82	1	5	10,70	25,13
173	5	2	7	6	4	5	0	10	1	4	4,40	2,95
174	1	0	6	1	3	2	9	1	7	1	3,10	3,11
175	1	11	3	4	4	4	1	11	4	6	4,90	3,54
176	10	1	4	1	10	5	15	12	6	5	6,90	4,68
177	10	3	11	0	8	2	3	2	1	2	4,20	3,94
178	0	5	3	11	3	14	6	6	0	2	5,00	4,55
179	6	1	5	4	6	14	0	2	7	3	4,80	3,97
180	6	20	10	0	6	4	4	0	2	2	5,40	5,97

LIITE E: REACT NATIVE -TESTISOVELLUKSEN SUORITTIMEN KÄYTTÖASTE: KUVAAJAT MITTAUSJAKSOITTAIN



LIITE F: SOVELLUKSEN AVAUTUMISAIKA

Ionic	Mittaus 1	Mittaus 2	Mittaus 3	Mittaus 4	Mittaus 5	Mittaus 6	Mittaus 7	Mittaus 8	Mittaus 9	Mittaus 10	Keskiarvo	Keskihajonta
Alku (s)	2,558	8,445	14,791	20,889	27,189	33,585	40,296	46,496	52,81	59,361		
Loppu (s)	5,497	11,695	17,82	24,371	30,271	37,093	43,496	49,706	56,69	62,632		
Aika (s)	2,939	3,250	3,029	3,482	3,082	3,508	3,200	3,210	3,880	3,271	3,285	0,276
React Native	Mittaus 1	Mittaus 2	Mittaus 3	Mittaus 4	Mittaus 5	Mittaus 6	Mittaus 7	Mittaus 8	Mittaus 9	Mittaus 10	Keskiarvo	Keskihajonta
Alku (s)	3,675	8,396	12,46	16,36	20,61	24,762	28,728	32,743	36,577	40,359		
Loppu (s)	5,044	9,526	13,507	17,457	21,777	25,876	29,897	33,825	37,69	41,561		
Aika (s)	1,369	1,13	1,047	1,097	1,167	1,114	1,169	1,082	1,113	1,202	1,149	0,090

LIITE G: KESKUSMUISTIN KÄYTTÖASTE

Keskusmuistin kokonaismäärä	2856 Mt											
Ionic	Mittaus 1	Mittaus 2	Mittaus 3	Mittaus 4	Mittaus 5	Mittaus 6	Mittaus 7	Mittaus 8	Mittaus 9	Mittaus 10	Keskiarvo	Käyttöaste
Alkutila (Mt)	2246	2179	2169	2177	2206	2184	2208	2203	2231	2212	2201,5	77,08 %
Korkein (Mt)	2258	2208	2230	2220	2258	2224	2245	2248	2267	2244	2240,2	78,44 %
Nousu (Mt)	12	29	61	43	52	40	37	45	36	32	38,7	1,36 %
React Native	Mittaus 1	Mittaus 2	Mittaus 3	Mittaus 4	Mittaus 5	Mittaus 6	Mittaus 7	Mittaus 8	Mittaus 9	Mittaus 10	Keskiarvo	Käyttöaste
Alkutila (Mt)	2178	2183	2165	2216	2219	2206	2206	2223	2221	2216	2203,3	77,15 %
Korkein (Mt)	2280	2228	2234	2279	2292	2325	2293	2314	2277	2275	2279,7	79,82 %
Nousu (Mt)	102	45	69	63	73	119	87	91	56	59	76,4	2,68 %